Univerza v Ljubljani
Fakulteta za računalništvo in informatiko

Manca Bizjak

Ovrednotenje praktične uporabnosti eksperimentalnih pristopov za zagotavljanje zasebnosti v oblaku

magistrsko delo

študijski program druge stopnje
računalništvo in informatika

Mentorica: doc. dr. Mojca Ciglarič

Ljubljana, 2016

Za pomoč, vodenje in nasvete se iz srca zahvaljujem mentorici doc. dr. Mojci Ciglarič. Prav tako se zahvaljujem mag. Alešu Černivcu za idejo, iz katere je zraslo to magistrsko delo, ter za pomoč in potrpežljivost.

Za moralno podporo tekom celotnega študija se zahvaljujem svoji družini, fantu Jaki in ostalim študijskim kolegom, še posebej Neži, Veroniki, Milošu in Žigi. Zaradi vas so bila moja študentska leta prijetnejša in mi bodo vedno ostala v lepem spominu.

Kazalo

Seznam uporabljenih kratic

kratica	angleško	slovensko
AJAX	asynchronous JavaScript and XML	asinhroni JavaScript in XML
BGV	Brakerski-Gentry-Vaikuntanathan cryptosystem	kriptosistem Brakerski-Gentry-Vaikuntanathan
CORS	cross-origin resource sharing	meddomensko deljenje virov
CSS	cascading style sheet	prekrivni slogi
DOM	document object model	objektni model dokumenta
HE	homomorphic encryption	homomorfna enkripcija
HTML	hypertext markup language	označevalni jezik za oblikovanje večpredstavnostnih dokumentov
HTTP	hypertext transfer protocol	protokol za izmenjavo večpredstavnostnih vsebin
IaaS	infrastructure as a service	infrastruktura kot storitev
JSON	JavaScript object notation	objektna notacija JavaScript
LLVM	low level virtual machine	nizkonivojska navidezna naprava
PaaS	platform as a service	računalniško okolje kot storitev
PHE	-	polna homomorfna enkripcija
RLWE	ring learning with errors	problem učenja z napakami nad kolobarji
SaaS	software as a service	programje kot storitev
TLS	transport layer security	sloj varnih vtičnic
TPM	trusted platform module	zaupanja vreden računalniški modul
URL	uniform resource locator	enotni naslov vira

Povzetek

Naslov: Ovrednotenje praktične uporabnosti eksperimentalnih pristopov za zagotavljanje zasebnosti v oblaku

Računanje z ohranjanjem zasebnosti je trenutno eno izmed najbolj obetavnih področij, ki ponuja pristope za zasebno obdelavo podatkov v računalniških oblakih. V pričujočem delu preučimo aktualne pristope za podporo zasebni obdelavi podatkov in se osredotočimo na implementacijo enega izmed novejših pristopov za računanje z ohranjanjem zasebnosti, polno homomorfno enkripcijo (PHE). Pri PHE je rezultat računskih operacij nad tajnopisi ob dekripciji enak kot pri izvedbi istega zaporedja operacij nad pripadajočimi čistopisi. Rezultat našega dela je preprosta spletna banka, ki s pomočjo PHE ohranja zasebnost bančnih transakcij. Za podporo PHE na strani odjemalca smo razvili dve arhitekturno različni programski rešitvi, ki sta združljivi z isto spletno aplikacijo. Prav tako smo definirali mere, na podlagi katerih smo ovrednotili praktično uporabnost naših implementacij v realnem okolju. Naši rezultati kažejo, da že pri trivialnih primerih uporabe PHE, kjer je cena procesiranja tajnopisov, enkripcije ali dekripcije zanemarljiva, naletimo na znatne količine režijskega dela zaradi komunikacije in inicializacije tako na strani strežnika kot tudi odjemalca. Prav tako oviro pri splošni uveljavitvi PHE v oblaku trenutno predstavlja tudi pomanjkanje visokonivojskih programskih konstruktov in razvijalcu prijaznih orodij. V obstoječi literaturi nismo našli navedb o nekaterih izmed omenjenih dejavnikov, ki omejujejo njeno širšo uveljavitev, saj naše delo obenem predstavlja enega izmed prvih primerov uporabe PHE preko spletne aplikacije.

Ključne besede: zagotavljanje zasebnosti, zasebno procesiranje, računalništvo v oblaku, polna homomorfna enkripcija.

Abstract

Title: Applicability evaluation of experimental approaches for preserving privacy in cloud

Among several proposed solutions for private processing in cloud computing, perhaps the most promising class of approaches is privacy-preserving computation. This thesis reviews existing approaches for private processing and demonstrates practical use of one such novel approach for privacy-preserving computation, fully homomorphic encryption (FHE). FHE allows arbitrary computations on ciphertexts and yields a result, which, when decrypted, is the same as if they were performed on corresponding plaintexts. We develop a simple web bank cloud application that uses FHE to preserve privacy of banking transactions. In order to support FHE client-side, we produce two architecturally different setups that can be used with the same web application. Furthermore, we evaluate and discuss their practical applicability in the cloud according to predefined metrics. Our results indicate that even for trivial use-cases, where performance of encrypted processing, encryption or decryption is not a limiting factor, the main factors preventing broader adoption of FHE at present are significant communication and initialization overheads both client and server-side, lack of support for several high level programmatic routines and lack of developer-friendly frameworks. To the best of our knowledge, some of these issues have not yet been addressed in the literature, as this thesis is one of the early attempts to bring FHE to the web.

Keywords: preserving privacy, private processing, cloud computing, fully homomorphic encryption.

Poglavje 1

Uvod

Trend selitve ogromnih količin podatkov v računalniške oblake odpira pomisleke o zasebnosti tako za posameznike kot tudi za podjetja [3, 19]. S stališča odjemalca računalniški oblak ni zaupanja vredno okolje, saj ima odjemalec omejen vpogled v ravnanje ponudnika oblačnih storitev z njegovimi podatki [42]. Ponudniki oblačnih storitev so sicer dolžni zagotoviti zakonsko predpisan nivo varnosti in zasebnosti, ki običajno vključuje uporabo dobro uveljavljenih mehanizmov za upravljanje identitet in za nadzor dostopa do računalniških virov. Kljub temu pa lahko na podlagi številnih incidentov iz prakse sklepamo, da uporaba zgolj teh pristopov odjemalca pogosto ne ščiti pred izgubo zasebnosti [26]. Prav tako zagotavljanje želenega nivoja zasebnosti v splošnem nima odločilne vloge pri izbiri ponudnika oblačnih storitev, saj se v praksi vsebina sporazumov o ravneh storitev (angl. *service level agreements, SLA*) praviloma osredotoča na kvaliteto storitev in performančne parametre [2, 40].

V znanstveni literaturi je trenutno zelo aktualno področje računanja z ohranjanjem zasebnosti (angl. *privacy preserving computation*), ki se osredotoča na zaupnost z namenom skrivanja odjemalčevih podatkov pred ponudniki oblačnih storitev [27, 36]. Računanje z ohranjanjem zasebnosti je sicer zgolj ena izmed možnih alternativ, kako pristopiti k ohranjanju zasebnosti v javnih računalniških oblakih, vendar pa v okviru le-tega najdemo več različnih, novejših in potencialno revolucionarnih pristopov s stališča uporabe v oblaku, kot so polna homomorfna enkripcija (šifriranje), iskljiva enkripcija in varno računanje med več subjekti.

Kljub dejstvu, da je v zadnjih letih računanje z ohranjanjem zasebnosti napre-

dovalo tako s teoretičnega kot tudi praktičnega vidika, pa je dejanska praktična uporaba omenjenih pristopov za reševanje problema zasebnosti v javnih računalniških oblakih še relativno mlado in nezrelo področje. Številni teoretični pristopi kljub razpoložljivosti programske podpore za njihovo realizacijo še nimajo uporabnih implementacij v oblaku [19, 42], zaradi česar je na omenjenem področju še veliko manevrskega prostora.

V pričujočem delu pregledamo trenutno najbolj aktualne pristope za zagotavljanje zasebnosti v oblaku, pri čemer se osredotočimo na podporo zasebni obdelavi podatkov. Posebno pozornost posvetimo pristopom, ki sodijo na področje računanja z ohranjanjem zasebnosti. Izmed slednjih izberemo po našem mnenju enega najbolj obetavnih pristopov, polno homomorfno enkripcijo, ki naslavlja enega zadnjih odprtih problemov specifičnih za varnost v oblaku, tj. notranjo grožnjo oziroma strah pred ponudnikom oblačnih storitev [36]. Uporabo PHE demonstriramo na primeru preproste oblačne aplikacije, ki simulira spletno banko. Pri tem predlagamo tri različne arhitekture programske rešitve, od katerih dve - pristop z razširitvijo za brskalnik Chrome in pristop z dodatnim posvečenim strežnikom - uspešno realiziramo in ju preizkusimo v realnem okolju.

Namen našega praktičnega dela je prikazati postopek, potreben za implementacijo podpore polni homomorfni enkripciji na primeru spletne aplikacije. Kljub zelo poenostavljenemu in s stališča uporabe polne homomorfne enkripcije skoraj trivialnemu primeru uporabe je bila implementacija težavna in dolgotrajna. Tekom razvoja smo naleteli na številne praktične prepreke, ki so nam onemogočile razvoj bolj sofisticirane aplikacijske logike. Na podlagi naše izkušnje z uporabo PHE v oblaku in na podlagi lastnih kriterijev in mer v delu ovrednotimo trenutni nivo njene praktične uporabnosti v oblačnih aplikacijah in izpostavimo glavne dejavnike, ki bi v bližnji prihodnosti utegnili predstavljati izzive pri njeni širši uveljavitvi na področju računalništva v oblaku.

Struktura preostanka pričujočega besedila je sledeča: v poglavju 2 razložimo osnovne pojme in predstavimo problematiko zasebnosti v računalniških oblakih, pri čemer se osredotočimo na zasebno obdelavo podatkov. Prav tako v poglavju 2 opišemo tudi metodologijo. Različne kategorije aktualnih pristopov za zagotavljanje zasebnosti med obdelavo podatkov in njihove najvidnejše predstavnike preučimo v poglavju 3. Kriterije, na podlagi katerih smo se odločili za implemen-

tacijo pristopa s polno homomorfno enkripcijo, in definicijo mer za vrednotenje praktične uporabnosti naših implementacij predstavimo v poglavju 4. Podrobnosti naših implementacij PHE na primeru preproste spletne banke opišemo v poglavju 5, razpravo o praktični uporabnosti PHE in naših implementacij na podlagi predhodno definiranih mer pa predstavimo v poglavju 6. Naše delo zaključimo s sklepnim poglavjem (poglavje 7), v katerem povzamemo glavne rezultate in izpostavimo nekaj ključnih odprtih vprašanj za nadaljnje praktično ter raziskovalno delo.

Poglavje 2

Pregled področja in metodologija

2.1 Računalniški oblak

Računalniški oblak je model porazdeljene računalniške arhitekture, v katerem odjemalci na zahtevo in na enostaven način preko omrežja dostopajo do množice deljenih računalniških virov. Ti računalniški viri, ki jih imenujemo tudi oblačni viri, so ponavadi omrežja, strežniki, pomnilniški viri, izvajalna okolja, aplikacije in storitve [29].

Za računalniške vire pravimo, da so deljeni, ker si več odjemalcev oblačnih storitev deli fizično infrastrukturo ponudnika. Deljenje virov omogoča uporaba *virtualizacije*, ki je ključni mehanizem, na katerem sloni oblačno računalništvo. Virtualizacija omogoča ločitev fizične infrastrukture v več navideznih (logičnih) enot, ki jih ponudnik oblačnih storitev da na razpolago odjemalcem. S pomočjo virtualizacije lahko na primer na danem računalniku zaganjamo eno ali več virtualnih naprav, ki simulirajo fizične računalnike z želenimi zmogljivostmi, na katere lahko namestimo poljubne operacijske sisteme [46].

Virtualizacija pa ne omogoča zgolj deljenja virov, ampak tudi njihovo fleksibilnost in skalabilnost. Odjemalec namreč lahko računalniške vire postavi hitro in s karseda malo interakcije s ponudnikom oblačnih storitev, prav tako pa jih po potrebi dodaja ali odstranjuje.

Nakup strojne ali programske opreme, skrb za njuno vzdrževanje in upravljanje ter učinkovita uporaba računalniških virov s tem niso več breme odjemalca, saj zanje skrbi ponudnik oblačnih storitev, odjemalcem pa računalniške vire ponudi v obliki storitve, ki jo obračuna glede na njihovo dejansko uporabo (t. i. model *pay-as-you-go*).

2.1.1 Odjemalci oblačnih storitev

Ena izmed ključnih praktičnih posledic uporabe oblaka je razbremenitev odjemalcev. Odjemalce oblačnih storitev označujemo kot *lahke odjemalce* (angl. *thin clients*), s čimer želimo poudariti, da gre za naprave z omejenimi računskimi viri in pomnilnikom. Zaradi omejenih virov lahki odjemalci na ponudnika oblačnih storitev preložijo čim večjo količino dela. Obenem pa so odjemalci ponavadi le kratek čas povezani z oblakom, običajno z namenom prenašanja in pregledovanja podatkov ali pa z namenom delegiranja dela nad podatki oblaku.

Kadar želimo poudariti odjemalce z omejenimi računskimi viri, običajno posebej izpostavimo mobilne naprave ali tablične računalnike, čeprav v vlogi lahkih odjemalcev nastopajo tudi prenosni in namizni računalniki.

2.1.2 Storitveni modeli

Uveljavljena definicija oblačnega računalništva [29] glede na nivo abstrakcije računalniških virov razlikuje med tremi storitvenimi modeli ponudnikov oblačnih storitev: *IaaS* (angl. *Infrastructure as a Service*, infrastruktura kot storitev), *PaaS* (angl. *Platform as a Service*, računalniško okolje kot storitev) in *SaaS* (angl. *Software as a Service*, programje kot storitev). Največ svobode pri upravljanju z oblačnimi viri imajo odjemalci v oblaku tipa IaaS, saj lahko sami izbirajo in upravljajo podporno infrastrukturo, na kateri nato lahko teče poljubna programska oprema. Odjemalci oblaka tipa PaaS možnosti upravljanja z oblačno infrastrukturo nimajo, lahko pa izkoristijo ponudnikove vire tako, da v oblaku objavijo svojo programsko opremo. V tem primeru ponudnik storitev v oblaku PaaS poskrbi za operacijski sistem, izvajalno okolje in programsko podporo, ki so potrebni za objavo odjemalčeve lastne programske opreme. Najvišji nivo abstrakcije pa predstavlja oblak tipa SaaS, kjer odjemalec nastopa zgolj v vlogi uporabnika storitve

oz. aplikacije, ki jo je ponudnik sam postavil v oblak.

2.1.3 Namestitveni modeli

Glede na razmerje med oblačno infrastrukturo in njenimi upravitelji ter odjemalci razlikujemo štiri namestitvene modele računalniških oblakov: *zasebni*, *skupinski*, *javni* in *hibridni* [29]. V zasebnem oblaku je lastnik oz. najemnik infrastrukture in njen upravitelj ponavadi organizacija, katere zaposleni so odjemalci oblačnih storitev. Storitve zasebnega oblaka so običajno na voljo odjemalcem znotraj organizacije. Podobno velja za skupinski oblak, le da si v tem oblačno infrastrukturo deli več organizacij s skupnimi interesi, tehnološkimi ali varnostnimi zahtevami. Nasprotno pa je lastnik infrastrukture v javnem oblaku organizacija, ki nudi oblačne storitve zunanjim odjemalcem. Njihove storitve so namenjene širši javnosti.

O hibridnem oblaku govorimo v primeru, če računalniški oblak sestavlja kombinacija (tipično dveh) ločenih, vendar s skupnimi tehnologijami povezanih oblakov - zasebni in javni, zasebni in skupinski ali javni in skupinski [46].

2.2 Problematika zasebnosti v računalniških oblakih

Ker odjemalec oblačnih storitev svojih podatkov ne hrani več na lastni napravi (lokalno), ampak se ti fizično nahajajo na infrastrukturi ponudnika oblačnih storitev, oblačno računalništvo s stališča odjemalca odpira pomisleke glede varnosti in zasebnosti. Vidik zasebnosti je še posebej izrazit v primeru selitve podatkov občutljive narave v javne računalniške oblake, kjer odjemalci niso lastniki oziroma najemniki fizične infrastrukture, in je posledično nivo vpogleda v delo ponudnika oblačnih storitev praviloma bistveno bolj omejen kot pri zasebnih oblakih.

2.2.1 Občutljivi podatki

Izraz občutljivi podatki bomo v nadaljevanju uporabili za skupno poimenovanje tipičnih podatkov, katerih razkritje bi njihovemu upravičenemu lastniku - osebi ali organizaciji - lahko prizadejalo osebno oziroma poslovno škodo.

Na ravni posameznikov kot občutljive smatramo na primer podatke o osebnem življenju in prepričanjih, o zdravstveni in kriminalni kartoteki ter vse podatke, ki se tičejo posameznika v vlogi stranke ali zaposlenega, in bi lahko vodili v njegovo identifikacijo.

S stališča organizacij pa so občutljive narave predvsem podatki o strankah, dobaviteljih in zaposlenih ter poslovne skrivnosti.

2.2.2 Trend uporabe oblakov

Uporaba tako zasebnih kot tudi javnih računalniških oblakov s strani posameznikov in organizacij iz leta v leto narašča [35]. Predvsem manjša podjetja si pogosto ne morejo privoščiti stroškov vzdrževanja lastne računalniške infrastrukture, čeprav je tudi pri večjih organizacijah selitev v računalniški oblak lahko strateška odločitev.

Letna analiza uporabe računalniških oblakov in z njimi povezanih varnostnih tveganj *Cloud Adoption and Risk Report* [38], izvedena ob koncu leta 2015, navaja, da naj bi več kot 15 odstotkov vseh datotek v računalniških oblakih vsebovalo občutljive podatke.

Zaradi izrazitega trenda naraščanja selitve podatkov in njihovega procesiranja v računalniške oblake, pri čemer pomembnen delež le-teh predstavljajo prav občutljivi podatki, je vprašanje zasebnosti tako za posameznike kot tudi za podjetja danes bolj aktualno kot kdaj koli prej.

2.2.3 Koncept zaupanja

V trenutku pisanja pričujočega besedila splošne tehnične rešitve, ki bi ponudniku oblačnih storitev omogočala netrivialno obdelavo (in ne zgolj shranjevanja) odjemalčevih podatkov, obenem pa odjemalcu zagotovila, da jih ponudnik ni zlorabil, ni.

Slednje za odjemalca pomeni, da mora ponudniku oblačnih storitev zaupati, da le-ta njegovih podatkov ne bo uporabil na način, ki ni bil predhodno dogovorjen [40]. Kljub vsem koristim, ki jih za odjemalca lahko prinese uporaba računalniškega oblaka, vključno z nižjimi stroški, večjo skalabilnostjo, fleksibilnostjo, zanesljivostjo in dosegljivostjo storitev, pa vzpostavitev zaupanja ponudniku oblačnih storitev ostaja velik izziv [39]. V praksi se danes o zaupanju v ponudnika

oblačnih storitev odločamo na podlagi podatkov o morebitnih preteklih varnostnih incidentih ponudnika in na podlagi sporazumov o ravneh storitve. Nič od omenjenega pa odjemalcu praviloma ne omogoča vpogleda v to, ali se ponudnik oblačnih storitev dogovorjenih vidikov glede varovanja zasebnosti zares drži.

V splošnem torej iščemo rešitve, ki karseda minimizirajo potrebni nivo zaupanja ponudniku oblačnih storitev, še posebej, če gre za javni računalniški oblak, ki je tako rekoč sinonim za zaupanja nevredno okolje.

2.2.4 Varnostne polemike, specifične za računalniške oblake

Računalništvo v oblaku je s svojimi posebnimi lastnostmi in arhitekturnimi značilnostmi z vidika varnosti uvedlo nekaj novih izzivov, ki jih pri drugih računalniških modelih praviloma ne srečujemo [46]. Literatura tipično navaja štiri za računalniški oblak specifične varnostne polemike, od katerih za prve tri v praksi že obstajajo močni in tehnično zreli protiukrepi [36]:

1. **V oblaku si računalniško infrastrukturo deli več odjemalcev, zato nekega odjemalca lahko ogroža drugi odjemalec.** Ponudnik oblačnih storitev lahko nepoštenim odjemalcem to prepreči ali pa vsaj močno oteži z uporabo upravljavcev virtualnih naprav in operacijskih sistemov, ki omogočajo dobro izolacijo procesov.

2. **Za dostop do podatkov v računalniških oblakih iz javnih omrežij se včasih namerno uporabljajo ne-varni protokoli in aplikacijski programski vmesniki.** Pred tem se lahko zavarujemo z uporabo protokolov, ki omogočajo zasebno komunikacijo, in z uporabo širše uveljavljenih rešitev za avtentikacijo, avtorizacijo ter nadzor dostopa.

3. **Ponudnik oblačnih storitev lahko odjemalčeve podatke nenamerno spremeni ali izgubi**. Nepooblaščene spremembe s strani ponudnika oblačnih storitev lahko zaznamo z uporabo mehanizmov za revidiranje (angl. *auditing*), pred izgubo podatkov pa se lahko zavarujemo z dosledno uporabo mehanizmov za ustvarjanje varnostnih kopij.

4. **Ponudnik oblačnih storitev, njegovi podizvajalci in zaposleni lahko dostopajo do odjemalčevih podatkov**. Ker imajo omenjene entitete dostop do odjemalčevih podatkov, jih lahko namerno ali pa nehote razkrijejo, kar v nadaljevanju označimo z izrazom *notranja grožnja*. Kot notranjo grožnjo bomo prav tako obravnavali kakršno koli uporabo odjemalčevih podatkov s strani omenjenih entitet, ki ni neposredno vezana na zagotavljanje storitve. Neposredni rezultat tako razkritja odjemalčevih podatkov kot tudi nepooblaščenega vpogleda v le-te za odjemalca pomeni izgubo zasebnosti.

Uresničitev katere koli izmed zgoraj opisanih varnostnih polemik lahko za odjemalca oblačnih storitev predstavlja poseg v njegovo zasebnost. Kljub temu pa je v času pisanja pričujočega besedila v praksi nerešen zgolj še problem notranje grožnje. Čeprav se praktična uvedba mehanizmov, ki naslavljajo prve tri varnostne polemike, lahko izkaže za težavno (in se posledično redko uporablja), v praksi za njihovo reševanje obstajajo zanesljive in preizkušeno dobre rešitve [36].

2.2.5 Notranja grožnja

Na nivo tveganja, ki ga za odjemalca predstavlja notranja grožnja, vpliva narava storitve, ki jo odjemalec koristi od ponudnika oblačnih storitev. Take storitve ponavadi zajemajo shranjevanje in obdelavo odjemalčevih podatkov.

Shranjevanje podatkov

Če odjemalec želi svoje podatke zgolj hraniti v oblaku, jih pred predajo v oblak lahko zakriptira. V primeru varnostnega incidenta, kjer pride do uhajanja njegovih podatkov, slednje za odjemalca samo po sebi ne pomeni izgube zasebnosti. Tajnopisi njegovih podatkov namreč za katero koli entiteto, ki nima v lasti dekripcijskega ključa - to seveda vključuje ponudnika oblačne storitve -, ne nosijo nobenega pomena. V primeru shranjevanja podatkov v oblaku je torej notranja grožnja trivialen problem, pred katerim se odjemalec lahko sam zavaruje z uporabo enkripcije [36].

Obdelava podatkov

Nemalokrat pa odjemalec svojih podatkov ne želi zgolj odložiti v oblak, ampak od ponudnika oblačnih storitev zahteva, da jih obdeluje (procesira) - na primer, nad njimi izvrši neko funkcijo ali pa po njih poizveduje. V primeru iz prejšnjega razdelka, kjer odjemalec zgolj shranjuje tajnopise svojih podatkov v računalniški oblak, jih lahko po tem, ko jih iz oblaka ponovno prenese k sebi in dekriptira, kvečjemu pri sebi lokalno obdela. Na tem mestu izpostavimo dva vidika, ki prikazujeta nepraktičnost opisanega scenarija:

1.) Bistvo številnih oblačnih storitev je prav v razbremenitvi odjemalca. Odjemalec morda niti nima razpoložljivih računalniških kapacitet (npr. procesorske moči ali pomnilnika), da bi bilo to izvedljivo.

2.) Tudi če procesorska moč in pomnilnik z vidika odjemalca nista problematična, je lahko količina podatkov, ki jo prenašamo iz oblaka, ogromna. Že zgolj prenos podatkov lahko zato traja zelo dolgo, nato pa je pred lokalnim procesiranjem na odjemalcu potrebna še dekripcija.

Očitno torej potrebujemo rešitev, ki procesiranja podatkov ne prelaga na odjemalca. Tu se seveda odpre vprašanje, kako aplikacijam zagotoviti normalno delovanje in obenem ohranjati zasebnost odjemalca. Aplikacije namreč obdelujejo čistopise odjemalčevih podatkov in ne tajnopisov. Slednje pomeni, da so odjemalčevi podatki tekom izvajanja oblačne aplikacije izpostavljeni tveganju razkritja, zaradi česar jih želimo ohraniti zaupne, torej jih skriti pred ponudnikom oblačne storitve.

Eden izmed možnih načinov, kako pristopiti k opisanemu problemu, je ta, da podatke v oblaku hranimo v kriptirani obliki, vendar s ponudnikom oblačne storitve delimo kriptografske ključe. Ko oblačna aplikacija potrebuje odjemalčeve podatke, jih lahko dekriptira, obdela in shrani rezultate, nato pa ponovno kriptira. Opisano pa je daleč od idealnega in nas pravzaprav vodi v protislovno situacijo, ko ponudniku oblačnih storitev še dodatno zaupamo upravljanje kriptografskih ključev. Po drugi strani pa se o notranji grožnji ne bi pogovarjali, če odjemalci ne bi imeli razloga za nezaupanje ponudniku oblačnih storitev.

Izkaže se torej, da je ohranjanje zasebnosti v primeru, ko oblak obdeluje odjemalčeve podatke, težek problem. Danes ponudniki oblačnih storitev naprednih

mehanizmov za zagotavljanje zasebnosti praviloma ne ponujajo [39]. Morebitna uporaba kakršnih koli naprednih mehanizmov za rešitev omenjenega problema je torej trenutno odvisna od tega, ali jo bodo razvijalci oblačnih aplikacij implementirali sami. Na odločitev ali bodo razvijalci v svoje aplikacije sami vgrajevali podporo mehanizmom za zasebno procesiranje neposredno vplivajo zahteve aplikacije po zmogljivosti in funkcionalnostih, prav tako pa tudi obstoj ustreznih programskih knjižnic za njihovo realizacijo.

2.3 Metodologija

Cilj našega praktičnega dela je razvoj spletne aplikacije, ki uporabnikom omogoča višjo raven zasebnosti med obdelavo podatkov, kot je za današnje aplikacije običajna. Za demonstracijo izdelamo preprosto spletno banko, ki izvaja bančne transakcije na način, ki ohranja zasebnost. V ta namen najprej preučimo pristope iz literature, ki so zaradi podpore zasebni obdelavi podatkov aktualni s stališča uporabe v računalniških oblakih. Za konkreten pristop, s katerim v aplikaciji podpremo zasebno obdelavo podatkov, se odločimo na podlagi lastnih kriterijev. Izbran pristop nato implementiramo s pomočjo razpoložljivih odprtokodnih programskih knjižnic in orodij. Spletno aplikacijo na koncu namestimo na strežnik v zasebni računalniški oblak tipa *IaaS*, v katerem je postavljena platforma OpenStack [33]. Na podlagi opažanj in meritev izbranih parametrov med uporabo spletne aplikacije nato ovrednotimo praktično uporabnost implementiranega pristopa. Ker strokovna literatura kot mero za vrednotenje praktične uporabnosti pristopov običajno navaja zgolj učinkovitost, bolj celovito ogrodje za vrednotenje praktične uporabnosti definiramo sami.

Poglavje 3

Zasebna obdelava podatkov v računalniških oblakih

V nadaljevanju bomo predstavili aktualne pristope, ki so bili predlagani kot možne alternative za podporo zasebni obdelavi podatkov v računalniških oblakih. Osredotočili se bomo na pristope, ki jih je mogoče uporabiti kot samostojne rešitve, čeprav bomo videli, da jih lahko med seboj tudi kombiniramo. Pri tem želimo izpostaviti temeljne koncepte vsakega izmed pristopov in poudariti posledice njihove uporabe v sistemih računalniških oblakov, zato se v opisih izognemo podrobnostim njihove realizacije.

Umestitev dela

Ne glede na to, kakšna je bila v nadaljevanju naša postavitev v okviru testiranja implementiranih rešitev, se v magistrskem delu namerno ne želimo usmeriti na noben konkreten storitveni model računalniškega oblaka. Zasebnost in z njo povezana tveganja v računalniškem oblaku, ki sta osrednja motivacija za pristope obravnavne v tem poglavju, sta namreč splošni težavi, aktualni v vseh oblačnih storitvenih modelih. Ne ozirajoč se na storitveni model pa želimo izmed namestitvenih modelov izpostaviti uporabno vrednost obravnavanih pristopov v javnih računalniških oblakih in se osredotočiti predvsem nanje, saj so tveganja za morebitne kršitve zasebnosti v le-teh običajno večja kot pri zasebnih, hibridnih ali

skupinskih oblakih.

3.1 Pristopi, ki temeljijo na podatkovni in informacijski varnosti ter zasebnosti

Skupna lastnost pristopov, ki temeljijo na zagotavljanju podatkovne oziroma informacijske varnosti in zasebnosti (angl. *data-centric* oz. *information-centric security and privacy*), je označevanje podatkovnih enot v skladu z drobnozrnatimi (angl. *fine-grained*) varnostnimi politikami [2, 39, 40]. V širšem smislu varnostne politike omogočajo nadzor nad pretakanjem informacij v računalniškem sistemu (angl. *information flow control*). Natančneje, z varnostnimi politikami lahko odjemalec sam izrecno predpiše, kako je z vidika varnosti in zasebnosti potrebno obravnavati neko podatkovno enoto, kdo in v katerih situacijah lahko do nje dostopa (jo dekriptira) ter kaj lahko v tem primeru z njo počne [40]. Pri tem je podatkovna enota ponavadi dokument ali skupina dokumentov, in jo, razen v primerih, ko jo obdelujejo pooblaščeni programi, v računalniškem oblaku shranjujemo v kriptirani obliki.

Z uvedbo varnostnih politik lahko torej odjemalec sam določi ne le katerim programom, aplikacijam ali uporabnikom naj se dovoli razkritje njegovih podatkov, ampak slednjim prav tako omeji njihovo uporabo po razkritju. Prav tako se dostopi do podatkovnih enot natančno beležijo z uporabo mehanizmov za revidiranje. Kljub temu da so njegovi podatki shranjeni v oblaku, ima torej odjemalec nad njimi precej večji nadzor, kot bi ga imel sicer, prav tako pa je obdelava njegovih podatkovnih enot s strani oblaka precej bolj transparentna.

Pristopi, ki se osredotočajo na podatkovno in informacijsko varnost in zasebnost, želijo v prvi vrsti zaščititi odjemalca tako pred namernim razkritjem njegovih podatkov s strani zlonamerne programske opreme kot tudi pred nenamernim razkritjem zaradi napak v programski opremi. Obenem pa želijo zagotoviti tudi izvzetje mehanizmov za varovanje odjemalčevih podatkov iz logike posameznih aplikacij, ki jih obdelujejo, ali pa vsaj abstrahiranje podrobnosti uveljavljanja varnostnih politik in revidiranja upoštevanja le-teh. Za slednje naj bi poskrbel ponudnik oblačih storitev, ki bi varovanje odjemalčevih podatkov lahko ponudil v obliki dodatne storitve. S tem pa bi bili prav tako razbremenjeni razvijalci

oblačnih aplikacij, ki obdelujejo občutljive podatke, saj bi se lahko osredotočili na razvoj poslovne logike aplikacij in bi v svojih aplikacijah zgolj uporabili obstoječe varnostne mehanizme.

3.2 Zaupanja vredno računanje

Zaupanja vredno računanje (angl. *trusted computing*) temelji na uporabi posebne programske opreme in strojne opreme, ki je odporna proti nepooblaščenim posegom.

Če o zaupanja vrednem računanju govorimo v kontekstu računalništva v oblaku, potem omenjena strojna oprema odjemalca varuje pred ponudnikom oblačnih storitev, saj se fizično nahaja v računalniškem oblaku [36]. Ta strojna oprema omogoča poseben način generiranja in shranjevanja kriptografskih ključev, pri katerem je vsak ključ logično povezan z vnaprej določeno množico programov. Samo ti programi lahko po potrebi dekriptirajo odjemalčeve podatke in jih uporabijo za poizvedovanje ali obdelavo.

Odjemalčevi podatki so torej v oblaku shranjeni v kriptirani obliki, vendar jih ponudnik oblačnih storitev v taki obliki ne more obdelovati. Lahko pa jih pred tem dekriptira, vendar le pod pogojem, da jih potrebuje program, ki je logično povezan z odjemalčevim kriptografskim ključem. Odjemalec mora pred tem svoj kriptografski ključ bodisi sam prenesti v računalniški oblak bodisi ga na novo zgenerirati s pomočjo omenjene strojne opreme, prav tako pa se mora s ponudnikom oblačnih storitev dogovoriti, katerim programom dovoli dostop do svojega dekripcijskega ključa in s tem do svojih podatkov.

Za strojno podporo zaupanja vrednemu računanju se v praksi uporabljajo zaupanja vredni moduli TPM (angl. *Trusted Platform Modules*), ki predstavljajo industrijski standard in so v praksi že dobro uveljavljeni. Moduli TPM zagotavljajo izvrševanje danega programskega sklada [36]. To pomeni, da poskrbijo za izvrševanje skupine izvršljivih datotek, ki smo jo določili sami, in ne morebitnih spremenjenih različic, ki bi jih utegnil podtakniti napadalec. Kljub temu pa moduli TPM ne nudijo zaščite pred ranljivostmi, ki izhajajo iz programske opreme, ki se izvršuje. Čeprav lahko programsko opremo zaščitimo pred nepooblaščenimi spremembami, nam moduli TPM namreč ne nudijo vpogleda v to, ali je programska

oprema, ki se izvršuje, zares zaupanja vredna.

Nasprotno pa se z varovanjem podatkov pred zlonamerno programsko opremo in pred napakami v programski kodi ukvarjajo pristopi iz prejšnjega razdelka. V literaturi predlagane rešitve za t. i. *varovanje podatkov kot storitev* (angl. *data protection as a service, DPaaS*) običajno sočasno uporabljajo pristope, ki temeljijo na podatkovni varnosti, ter module TPM [39].

3.3 Računanje z ohranjanjem zasebnosti

V primeru uporabe pristopov iz razdelkov 3.1 in 3.2 ima strežnik med procesiranjem še vedno vpogled v zasebne podatke odjemalca. Slednje lahko predstavlja težavo za različne primere uporabe, v katerih mora oblačna aplikacija zadostiti določenim zahtevam po zasebnosti [27]. Omenjen problem lahko naslovimo z uporabo tehnik za računanje z ohranjanjem zasebnosti (angl. *privacy-preserving computation*), pri katerih operacija oz. funkcija, ki se izvede nad podatki, v splošnem ne razkriva zasebnih podatkov (vključno z vhodi in izhodi). Entiteta, ki procesira odjemalčeve podatke, se torej na podlagi izvrševanja aplikacijske kode praviloma ne more naučiti ničesar o podatkih samih. Na ta način dosežemo višji nivo zasebnosti kot pri tradicionalnih aplikacijah, kjer strežnik procesira čistopise.

Vsi pristopi za računanje z ohranjanjem zasebnosti se močno naslanjajo na uporabo kriptografije kot temeljnega orodja za podporo zasebnosti v računalniških sistemih. Kljub temu da je praktična implementacija kriptografskih primitivov za vse pristope, ki jih bomo predstavili v nadaljevanju, že sama po sebi kompleksna, pa to ni edina težava, s katero se srečujemo pri njihovi potencialni uporabi v oblačnih aplikacijah. Pri uveljavitvi omenjenih pristopov v praksi je namreč pogosto večja ovira pomanjkanje uporabnikom in razvijalcem prijaznih implementacij in orodij [27].

V nadaljevanju si bomo ogledali nekaj pristopov, ki lahko na implementacijski ravni podprejo različne funkcionalnosti - zasebno poizvedovanje, procesiranje in deljenje podatkov.

3.3.1 Enkripcija z ohranjanjem vrstnega reda

Enkripcija z ohranjanjem vrstnega reda (angl. *order preserving encryption*) je deterministična simetrična enkripcija, ki ohranja numerično ureditev tajnopisov glede na pripadajoče čistopise. Zaradi ohranitve relacije urejenosti omogoča izvajanje primerjalnih operacij nad tajnopisi in enostavno izvajanje intervalnih poizvedb (npr. poizvedba po vseh vrsticah tabele v podatkovni bazi, kjer so vrednosti danega stolpca med a in b), saj potrebujemo zgolj tajnopisa robnih točk intervala [5]. Enkripcijske sheme prav tako podpirajo binarno iskanje po tajnopisih, s čimer dosegajo logaritemsko časovno zahtevnost iskanja.

Ker lahko ob uporabi enkripcijskih shem z ohranjanjem vrstnega reda operatorja manjši in večji uporabimo neposredno nad tajnopisi, so slednje sheme aktualne predvsem na področju zasebnih podatkovnih baz, kjer bi sicer morali tajnopise pred poizvedovanjem dekriptirati.

Enkripcijske sheme z ohranjanjem vrstnega reda se razlikujejo po naboru podatkovnih tipov, ki jih podpirajo, po morebitnih implementacijah funkcij iskanja največjega ali najmanjšega elementa, štetja, sortiranja in grupiranja [1]. Vsem tovrstnim enkripcijskim shemam pa je skupno, da strežniku omogočajo vpogled v rezultat operacij nad tajnopisi. To pomeni, da se strežnik, kljub temu da ne pozna čistopisov, zaveda, ali je dani tajnopis večji oz. manjši od drugega tajnopisa, posledično pa ima tudi informacijo o relaciji med pripadajočimi čistopisi. S pomočjo rezultatov poizvedb se torej strežnik lahko postopoma uči in pridobiva omejeno znanje o odjemalčevih podatkih, kljub temu da se ne zaveda njihovega pomena.

3.3.2 Iskljiva enkripcija

Podobno kot enkripcija z ohranjanjem vrstnega reda je tudi *iskljiva* enkripcija (angl. *searchable encryption*) aktualna predvsem zaradi njene potencialne uporabne vrednosti pri zasebnem poizvedovanju po podatkovnih bazah.

Iskljive enkripcijske sheme omogočajo iskanje ključnih besed nad tajnopisi skozi primerjavo enakosti dveh tajnopisov, s čimer lahko podpremo iskanje dane vrednosti v kriptirani podatkovni bazi. Lahko bi rekli, da je iskljiva enkripcija pravzaprav poseben primer enkripcije z ohranjanjem vrstnega reda, čeprav za razliko od slednjih obstajajo varne implementacije tako simetričnih kot tudi asimetričnih

iskljivih shem.

Iskljiva enkripcija je še posebej aktualna za primere, ko podatke shranjujemo v zaupanja nevredno okolje (npr. javni računalniški oblak), zaradi česar jih sami predhodno zakriptiramo. Omogoča nam, da zaobidemo prenašanje podatkov iz zaupanja nevrednega okolja k odjemalcu, dekripcijo celotne podatkovne baze in iskanja po čistopisih. Prav tako pa s tem tudi drugim odjemalcem omogočimo, da lahko po podatkovni bazi poizvedujejo brez vpogleda v čistopise.

Prav tako kot enkripcijske sheme z ohranjanjem vrstnega reda tudi iskljive enkripcijske sheme omogočajo intervalne poizvedbe po tajnopisih, ne pa tudi binarnega iskanja. Zato je potrebno vsakič, ko želimo iskati po kriptirani podatkovni bazi, tajnopis ključne besede, po kateri poizvedujemo, primerjati z vsemi tajnopisi. Povedano drugače, za vsako poizvedbo moramo pregledati celotno podatkovno bazo, preden dobimo popoln rezultat. Slednje je velika pomanjkljivost velike večine tako simetričnih kot tudi asimetričnih iskljivih enkripcijskih shem, saj pomeni linearno časovno zahtevnost iskanja, za razliko od enkripcijskih shem z ohranjanjem vrstnega reda, ki imajo zaradi relacije urejenosti logaritemsko časovno zahtevnost. Podpora iskljivi enkripciji nam s tem onemogoča uporabo indeksiranja kot ključnega mehanizma za doseganje učinkovitejših poizvedb, zaradi česar so iskljive enkripcijske sheme v praksi manj priljubljene od tistih z ohranjanjem vrstnega reda [27].

3.3.3 Homomorfna enkripcija

Čeprav enkripcija z ohranjanjem vrstnega reda in iskljiva enkripcija omogočata poizvedovanje po tajnopisih, ne nudita podpore poljubnemu procesiranju tajnopisov. Procesiranje tajnopisov v pravem pomenu, torej izvajanje funkcij neposredno nad tajnopisi, obravnava homomorfna enkripcija (HE). Rezultat homomorfnega procesiranja je tajnopis, ki ob dekripciji da rezultat, ki odraža izvajanje neke (ne nujno enake) funkcije neposredno nad pripadajočimi čistopisi.

O homomorfnih enkripcijskih shemah se je govorilo že dolgo pred pojavom oblačnega računalništva, tako da na področju kriptografije pravzaprav niso novost; dobro uveljavljene enkripcijske sheme RSA brez bitnega zapolnjevanja, El Gamal in Pallierjeva shema so homomorfne [30, 41]. Vendar pa moramo biti pri klasifikaciji homomorfnih enkripcijskih shem natančni, saj razlikujemo *delno*, *polno*

in *nekoliko* homomorfno enkripcijo, ki jih predstavimo v nadaljevanju.

Delna homomorfna enkripcija

Delno homomorfne (angl. *partially homomorphic*) enkripcijske sheme so bodisi multiplikativno bodisi aditivno homomorfne. Enkripcijski shemi RSA in El Gamal sta multiplikativno homomorfni, pri obeh pa velja, da rezultat množenja tajnopisov ob dekripciji da enak rezultat, kot če bi zmnožili pripadajoča čistopisa. Pri Pallierjevi enkripcijski shemi pa množenje tajnopisov da rezultat, ki ob dekripciji zrcali seštevanje pripadajočih čistopisov, in jo zato uvrščamo med aditivne homomorfne enkripcijske sheme.

Zgolj homomorfno seštevanje ali množenje pa ne zadostuje za realizacijo poljubnih funkcij nad tajnopisi.

Polna homomorfna enkripcija

Velik preboj na področju kriptografije je predstavljal predlog prve *polne* homomorfne (angl. *fully homomorphic*) enkripcijske sheme, tj. sheme, ki je hkrati multiplikativno in aditivno homomorfna (Gentry, 2009 [12]). Kombinacija operacij seštevanja in množenja namreč predstavlja funkcijsko poln nabor, kar pomeni, da lahko z njima v praksi realiziramo katero koli funkcijo. Polna homomorfna enkripcija (PHE) s tem omogoča realizacijo poljubnih funkcij nad tajnopisi brez potrebe po predhodni dekripciji podatkov s strani entitete, ki procesira podatke.

Slednje ima velike praktične posledice za računalniške oblake, saj pomeni, da lahko oblak v teoriji obdeluje tajnopise odjemalčevih podatkov, kot da bi bili ti čistopisi, kar ohranja zasebnost odjemalčevih podatkov ne le med shranjevanjem v oblaku, ampak tudi med procesiranjem. Omenjena lastnost je razlog, zaradi katerega je bila v zadnjih nekaj letih PHE deležna velike pozornosti. Tako odjemalčevi podatki kot tudi rezultati procesiranja le-teh namreč ostanejo skriti pred ponudnikom oblačnih storitev.

Zavedati pa se moramo, da ima tak pristop k ohranjanju zasebnosti pomembne posledice za programe, ki izvajajo homomorfno procesiranje. Ti namreč ne morejo več implementirati odločanja na podlagi vrednosti podatka, saj poznajo samo njen tajnopis. Ista lastnost lahko programerjem še dodatno oteži razhroščevanje

programov, saj nimajo vplogleda v vrednosti spremenljivk, nad katerimi se izvaja homomorfno procesiranje [36, 39].

Odkar je Gentryjeva shema za PHE leta 2012 dobila svojo prvo praktično implementacijo, je bilo razvitih in implementiranih še več polno homomorfnih enkripcijskih shem. Vendar pa je skupni imenovalec vseh zelo nizka hitrost homomorfnih operacij v primerjavi s hitrostjo ekvivalentnih operacij nad čistopisi. Čeprav se je do danes hitrost homomorfnega procesiranja v nekaterih implementacijah povečala za več velikostnih redov, je še vedno najbolj omejujoč faktor, ki preprečuje širšo uveljavitev PHE v praksi [30]. Na tem mestu omenimo še dejstvo, da so polne homomorfne enkripcijske sheme asimetrične. V praksi asimetričnih enkripcijskih shem praviloma ne uporabljamo za enkripcijo večjih količin podatkov, saj so zaradi učinkovitosti nepraktične, tudi če niso homomorfne.

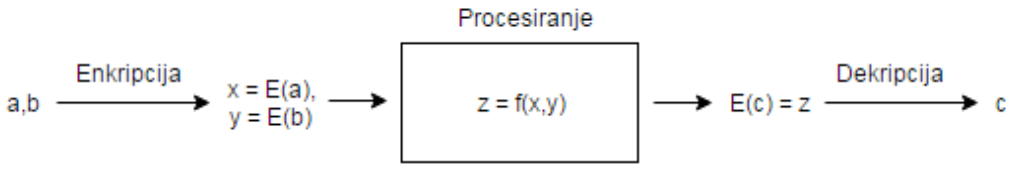
Shematičen prikaz homomorfnega procesiranja ter lastnosti aditivnega, multiplikativnega in polnega homomorfizma lahko vidimo na sliki 3.1.

Nekoliko homomorfna enkripcija

Poleg delne in polne homomorfne enkripcije v literaturi srečujemo tudi *nekoliko* homomorfno (angl. *somewhat homomorphic*) enkripcijo. Nekoliko homomorfne enkripcijske sheme omogočajo omejeno število homomorfnih operacij, ki so lahko seštevanja in množenja, s tem pa realizacijo omejenega nabora funkcij nad tajnopisi [30]. Lahko bi rekli, da so tovrstne enkripcijske sheme kompromis med delno in polno homomorfno enkripcijo, saj omogočajo tako seštevanje kot tudi množenje, vendar le določeno in vnaprej znano število enih ali drugih.

Nekoliko homomorfne enkripcijske sheme za doseganje varnosti v tajnopise vnašajo *šum*. Vsaka homomorfna operacija popači, tj. vnese šum v dani tajnopis. Z naraščanjem števila homomorfnih operacij nad tajnopisom postane šum tako velik, da tajnopisa ni več mogoče pravilno dekriptirati, kar pomeni, da ob dekripciji dobimo napačen rezultat [22]. To je razlog, da nekoliko homomorfne enkripcijske sheme podpirajo omejeno število homomorfnih seštevanj ali množenj.

Čeprav lahko nekoliko homomorfne enkripcijske sheme nastopajo samostojno, se v praksi pogosto uporabljajo kot gradniki za realizacijo PHE.



(a) Shematičen prikaz homomorfnega procesiranja

Velja: $c = a + b$

(b) Aditivni homomorfizem

Velja: $c = a \cdot b$

(b) Multiplikativni homomorfizem

Velja: $c = f(a, b)$

(b) Polni homomorfizem

Slika 3.1: (a) Vizualizacija lastnosti homomorfnih enkripcijskih shem. Tajnopis podatka x je na sliki prikazan kot $E(x)$. Ključno je, da so vhodi in izhod funkcij $f(x, y)$ tajnopisi in da postopek procesiranja ne vključuje dekripcije. Funkcije f imajo lahko v splošnem poljubno število vhodov. Glede na vrsto homomorfizma velja zveza (b) za aditivne, (c) za multiplikativne in (d) za polne homomorfne enkripcijske sheme.

Samozagon nekoliko homomorfnih enkripcijskih shem Vse današnje polne homomorfne enkripcijske sheme neomejeno število seštevanj in množenj nad tajnopisi dosežejo tako, da za osnovo vzamejo nekoliko homomorfno enkripcijsko shemo in jo pretvorijo v polno homomorfno s postopkom *samozagona* (angl. *bootstrapping*) [23]. Samozagon omogoča osveževanje tajnopisov, katerega rezultat je zmanjšanje šuma. Posledično lahko po samozagonu nadaljujemo z izvajanjem homomorfnih operacij, dokler šum v tajnopisu ne postane prevelik, nato pa ga ponovno osvežimo. Postopku osveževanja pravimo tudi *rekripcija* (angl. *recryption*). Gre za računsko zelo zahteven postopek, zaradi katerega so polne homomorfne enkripcijske sheme precej manj učinkovite kot nekoliko homomorfne enkripcijske sheme.

3.3.4 Varno računanje med več subjekti

Vsi do sedaj opisani pristopi za računanje z ohranjanjem zasebnosti predvidevajo prisotnost zgolj enega subjekta - odjemalca. Varno računanje med več subjekti (angl. *secure multi-party computation*) pa je bolj splošen pristop, ki predvideva sodelovanje več subjektov. Ti subjekti si med seboj ne zaupajo in zato želijo skriti svoje vhodne podatke pred drugimi, obenem pa izračunati vrednost skupne funkcije s prispevki vsakega subjekta posebej. V primeru, da varnega računanja med več subjekti ne bi uporabljali, bi potrebovali entiteto, ki bi poznala vhodne podatke vseh udeleženih subjektov [10]. Ker se v zaupanja nevrednih okoljih javnih računalniških oblakov pogosto ne želimo zanašati na zaupanje v ponudnika oblačnih storitev ali tretji entiteti, varna komunikacija med več subjekti predstavlja alternativo, ki zagotovljeno ohranja zasebnost vhodnih podatkov vsakega subjekta.

Varno računanje med množico subjektov $S_1, S_2, ...S_n$ torej udeleženim subjektom omogoča izračun funkcije $f(x_1, x_2, ...x_n)$, pri čemer je x_i vhodni podatek subjekta S_i in vsak od subjektov Si prispeva natanko en vhodni podatek, ki je neodvisen od vhodnih podatkov vseh ostalih subjektov [14]. Bistvo varnega računanja med več subjekti je v tem, da subjekt S_i nima vpogleda v vhodne podatke ostalih udeleženih subjektov (torej za x_j, če $j \neq i$) in tako vhodni podatki vsakega subjekta ostanejo skriti pred vsemi ostalimi. Povedano drugače, vse kar se lahko subjekt S_i nauči o vhodnih podatkih ostalih subjektov, je to, kar lahko o njih sklepa iz izhoda funkcije in iz lastnega vhoda [27].

Varna komunikacija med več subjekti ne ohranja zasebnosti izhoda funkcije (slednjega ugotovijo vsi udeleženi subjekti) in se ne ukvarja z vprašanjem, koliko se lahko iz izhoda funkcije in lastnih vhodov dani subjekt nauči o vhodih drugih subjektov. Literatura je zato ne uvršča vedno med pristope za računanje z ohranjanjem zasebnosti v pravem pomenu, ampak jo včasih obravnava zgolj kot pristop za izboljšanje zasebnosti.

3.3.5 Zasebno pridobivanje informacij

Zasebno pridobivanje informacij (angl. *private information retrieval*) je pristop, ki na eni strani odjemalcu omogoča zasebno poizvedovanje po podatkovni bazi,

na drugi strani pa ohranja tudi zasebnost lastnika podatkovne baze. Odjemalec - v tem primeru uporabnik podatkovne baze - pri tem želi pridobiti podatek x_i iz zbirke podatkov $x = x_1, x_2, ..., x_i, ..., x_n$, ne da bi lastniku podatkovne baze razkril x_i. Da pa se ohranja tudi zasebnost lastnika podatkovne baze, obenem odjemalcu ne želimo poslati celotne podatkovne zbirke x [10].

V primeru, da se podatkovna baza nahaja v računalniškem oblaku, zasebno pridobivanje informacij tako uporabnike podatkovne baze kot tudi njenega lastnika varuje pred ponudnikom oblačnih storitev [25].

Zasebno pridobivanje informacij se od iskljive enkripcije in enkripcije z ohranjanjem vrstnega reda, opisanih v razdelkih 3.3.1 in 3.3.2 razlikuje v tem, da zasebnosti poizvedb ne zagotavlja s pomočjo poizvedovanja po tajnopisih. Namesto specializiranih enkripcijskih shem za poizvedovanje po tajnopisih zasebno pridobivanje informacij sloni na deljenju skrivnosti, ki ga implementira z mehanizmom za varno računanje med več subjekti. Zaradi slednjega ga literatura včasih klasificira kot poseben primer varnega računanja med več subjekti.

Podobno kot pri ostalih obravnavanih pristopih za zagotavljanje višje ravni zasebnosti v oblaku je tudi pri zasebnem pridobivanju informacij problematična učinkovitost. Zmogljivost in slaba skalabilnost sta dve veliki pomanjkljivosti shem za zasebno pridobivanje informacij. Njihova velika prednost pa je ta, da jih je praviloma mogoče enostavno paralelizirati in s tem pohitriti [9, 10], kar je s praktično neomejenimi računskimi viri v računalniških oblakih enostavno dosegljivo.

Poglavje 4

Izbira pristopa in vrednotenje

V prejšnjem poglavju smo predstavili nekaj s stališča računalništva v oblaku trenutno najbolj aktualnih pristopov za podporo višji ravni zasebnosti v netrivialnih scenarijih. Ker pa je bil eden izmed ciljev našega dela izbrati nekaj pristopov za realizacijo in jih preizkusiti v praksi, je bil pomemben vidik tudi odločitev za konkretne pristope. Izmed vseh kandidatov, opisanih v poglavju 3, smo se odločili za praktično realizacijo polne homomorfne enkripcije. V nadaljevanju predstavimo kriterije, na podlagi katerih smo se odločili zanjo, in definiramo mere, s katerimi smo po zaključku razvoja ovrednotili naše implementacije s stališča praktične uporabnosti.

4.1 Kriteriji za izbiro pristopa

Pri izbiri pristopa smo v navedenem zaporedju pregledali štiri za nas pomembne vidike, na podlagi katerih smo sproti izločali kandidate za realizacijo:

1.) **Možnost realizacije pristopa izključno s programsko opremo**
Ker na razpolago nismo imeli nobene specializirane strojne opreme, smo med kandidati iskali take, ki jih je mogoče realizirati izključno s programsko opremo.

V tem koraku smo izločili *zaupanja vredno računanje* in *pristope, ki temeljijo na podatkovni in informacijski varnosti in zasebnosti*, saj so slednji praviloma podprti s strani modulov TPM ali z drugimi vrstami specializirane

strojne opreme.

Omejitev na realizacijo zgolj s programsko opremo je zožila naš nabor preostalih kandidatov samo na pristope iz kategorije računanja z ohranjanjem zasebnosti.

2.) **Podpora aplikacijskim funkcionalnostim**
Na tej točki smo se sami odločili o funkcionalnostih, ki smo jih želeli demonstrirati v naši spletni aplikaciji. Odločili smo se za implementacijo aplikacije v finančni domeni, ki bi omogočala procesiranje podatkov na način, ki ohranja zasebnost, pri čemer še nismo natančno definirali poslovne logike.

Odločitev za podporo zasebnemu procesiranju je skrčila nabor kandidatov na *homomorfno enkripcijo* in *varno računanje med več subjekti*. Ker imajo *zasebno pridobivanje informacij*, *enkripcija z ohranjanjem vrstnega reda* in *iskljiva enkripcija* uporabno vrednost zgolj na področju zasebnega poizvedovanja po podatkovnih bazah, niso izpolnili naših zahtev.

3.) **Splošnost pristopa**
Med kandidati sta ostala še *varno računanje med več subjekti* in *homomorfna enkripcija*, ki je pravzaprav skupina pristopov. Kot smo videli v razdelku 3.3.3, ločimo več tipov homomorfnih enkripcijskih shem, ki se razlikujejo po tipu in številu homomorfnih operacij, ki jih podpirajo, s tem pa tudi po naboru funkcij, ki jih z njimi lahko realiziramo. Ker prvotno nismo imeli natančno definirane poslovne logike ciljne aplikacije, smo izločili *delno* in *nekoliko homomorfno enkripcijo* ter se odločili za *polno homomorfno enkripcijo*, saj je z njo mogoče realizirati poljubne funkcije.

4.) **Obstoječe implementacije**
Po prejšnjem koraku izločanja sta nam ostala le še *varno računanje med več subjekti* in *polna homomorfna enkripcija*. Za implementacijo obeh je bilo na voljo več programskih knjižnic, vendar smo se nenazadnje odločili zgolj za realizacijo polne homomorfne enkripcije. Kljub temu da danes noben od preostalih dveh pristopov ni širše uveljavljen v računalniških oblakih, ima sodeč po literaturi varno računanje med več subjekti bistveno več praktičnih implementacij. Nekatere izmed njih so bile testirane tudi v realnih okoljih računalniških oblakov [4].

Po drugi strani pa je pristop s polno homomorfno enkripcijo novejši in posledično manj zrel. Opazili smo, da večina obstoječih implementacij iz literature ne predstavlja celovitih rešitev, saj se praviloma osredotočajo zgolj na homomorfno procesiranje, ki se odvija na strani strežnika v računalniškem oblaku. Odjemalci in njihov vidik uporabe oblačnih storitev, ki slonijo na polni homomorfni enkripciji, pa je v tovrstnih rešitvah pogosto prezrt.

4.2 Vrednotenje praktične uporabnosti

Čeprav bomo v poglavju 6 na kratko komentirali praktično uporabnost PHE kot pristopa, se bomo pri vrednotenju osredotočili predvsem na naše konkretne implementacije in uporabljena programska orodja, s katerimi so le-te zgrajene. V nadaljevanju torej navajamo mere, s pomočjo katerih bomo ovrednotili praktično uporabnost lastnih implementacij.

4.2.1 Produkcijska zrelost

Produkcijska zrelost je kvalitativna mera, ki jo bomo definirali kot sestavljeno mero. Obsegala bo funkcionalno dovršenost programskih orodij za implementacijo pristopov in obstoj morebitnih standardov za njihovo uporabo v računalniških oblakih. Produkcijska zrelost torej v veliki meri sloni na konkretnih programskih orodjih, ki jih uporabljamo za realizacijo pristopa.

V trenutku pisanja tega besedila je PHE neuveljavljen pristop, ki ni predmet nobenega nam znanega računalniškega standarda. Zato bomo produkcijsko zrelost naslovili zgolj na nivoju programskih orodij za njeno realizacijo, in sicer pregledali bomo trenutno stanje uporabljenih programskih orodij, katere v praksi zaželjene lastnosti imajo in kaj so njihove omejitve.

4.2.2 Transparentnost za uporabnika

Za praktično uporabo katerega koli pristopa oz. njegovih implementacij se nam zdi pomembno, da je njihovo delovanje za uporabnika kar se da transparentno. To pomeni, da preferiramo rešitve, ki za svoje predvideno delovanje zahtevajo čim manjšo vpletenost uporabnika, in so čim bolj učinkovite.

Nivo transparentnosti naših implementacij za uporabnika bomo merili posredno, in sicer preko dveh količin:

- števila operacij, ki jih mora uporabnik ročno izvesti v spletni aplikaciji v okviru neke akcije, potrebne izključno za podporo PHE in
- odzivnega časa. Odzivni čas obravnavamo kot čas, ki preteče od trenutka, ko uporabnik v spletni aplikaciji sproži neko akcijo, do trenutka, ko dobi povratno informacijo o izvedbi te akcije. Pri tem se osredotočimo samo na akcije, ki so nujne za podporo PHE.

4.2.3 Poraba računalniških virov v oblaku

Spremljali bomo tudi učinek, ki ga ima strežniški del naše aplikacije na računalniški oblak med polnim homomorfnim procesiranjem. Izmerili bomo naslednje parametre:

- obremenitev procesorja,
- porabo delovnega pomnilnika in
- porabo prostora na disku.

Kljub temu da dodatna obremenitev oblačnih virov na račun implementacije PHE ni tako kritična (računalniški oblak je nenazadnje okolje s skorajda neomejenimi računskimi viri), pa se neposredno odraža v ceni, ki jo bo zaračunal ponudnik oblačnih storitev. Dokler se mehanizmi za uporabo PHE v oblaku ne bodo uveljavili, bodo namreč najemniki oblačnih storitev tisti, ki bodo plačevali povišane stroške zaradi njihove uporabe.

4.2.4 Količina omrežnega prometa

Za praktično uporabo je pomemben tudi vidik komunikacije med odjemalcem in strežnikom v računalniškem oblaku. Prenos velike količine podatkov po počasni omrežni povezavi namreč povečuje odzivne čase v spletni aplikaciji, s tem pa zmanjšuje transparentnost za uporabnika.

Izmerili bomo količino dodatnega prometa, ki se bo na račun PHE prenašal med odjemalcem in strežnikom v različnih primerih uporabe aplikacije.

Poglavje 5

Implementacija polne homomorfne enkripcije v spletni aplikaciji

Osrednji rezultat našega dela je skupina programskih rešitev, ki preprosti spletni aplikaciji za bančništvo omogočajo uporabo polne homomorfne enkripcije za izvedbo bančnih transakcij. Izvajanje bančnih transakcij v naši postavitvi zato omogoča večjo zasebnost, kot je za tovrstne aplikacije danes običajno. V primeru, da je strežnik banke nameščen v računalniškem oblaku, naša aplikacija občutljive podatke o bančnih transakcijah in zneskih na računih strank banke varuje pred ponudnikom oblačnih storitev. Čeprav spletna aplikacija v praksi pogosto sovpada s pojmom oblačne aplikacije, je v našem primeru spletna aplikacija le gradnik oblačne aplikacije, ki implementira poslovno logiko banke, se pa obenem naslanja tudi na naše nizkonivojske programske rešitve za podporo PHE.

V tem poglavju v razdelku 5.1 najprej predstavimo spletno aplikacijo banke, nato pa se v razdelku 5.2 posvetimo podrobnostim implementacije nizkonivojske kode za podporo PHE tako na strani spletne banke (strežnika) kot tudi na napravi končnega uporabnika (odjemalca). Opise predlaganih rešitev na strani odjemalca, ki jih lahko uporabljamo z isto aplikacijo spletne banke, in se poslužujejo omenjene nizkonivojske kode, podajamo v razdelkih 5.3 do 5.5.

5.1 Spletna aplikacija banke

Aplikacije s področja financ in bančništva ne glede na kompleksnost poslovne logike praviloma zahtevajo obdelavo podatkov o bančnih transakcijah in zneskih na transakcijskih računih. Tudi v naši spletni aplikaciji, ki ima sicer zelo preprosto poslovno logiko, smo se zato osredotočili predvsem na slednje. V nadaljevanju podajamo nekaj funkcionalnih in tehničnih lastnosti implementirane spletne aplikacije.

5.1.1 Izhodišča in predpostavke

Spletno aplikacijo smo izdelali v programskem jeziku Python s pomočjo ogrodja za izgradnjo spletnih aplikacij Flask [20]. Za razliko od realnih spletnih bank naša aplikacija ne implementira avtentikacije odjemalcev s pomočjo digitalnih potrdil in ne uporablja protokola TLS za prenos prometa HTTP. Predpostavljamo namreč, da je uporaba slednjih v realnih spletnih bankah samoumevna.

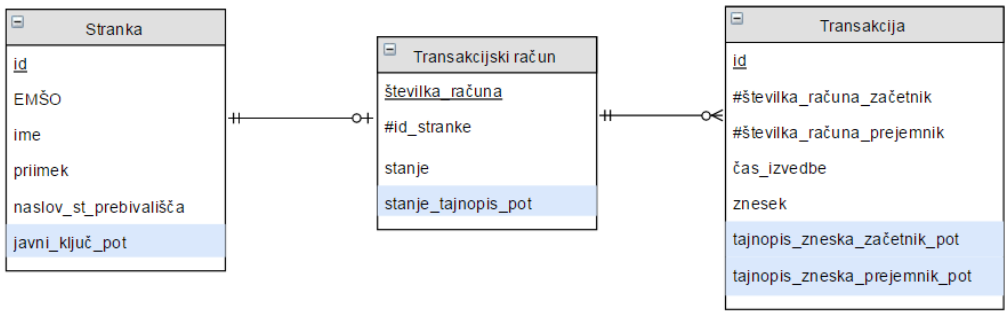
5.1.2 Podatkovni model

Poenostavitvam aplikacijske logike naše spletne banke je sledil tudi njen podatkovni model. Definirali smo zgolj podatkovne modele za stranko, transakcijski račun in transakcijo. Logični model podatkovne baze z definiranimi entitetnimi tipi in razmerji med njimi prikazuje slika 5.1.

Iz slike 5.1 je razvidno, da smo poleg lastnih atributov entitetnim tipom dodali še atribute za podporo PHE. Slednji zaradi splošnosti in združljivosti z nizkonivojskimi programskimi rešitvami za podporo PHE v programskem jeziku C++ (opisanimi v nadaljevanju v razdelku 5.2.3) predstavljajo poti do datotek s kriptografskim materialom - ključev in tajnopisov.

Stranka

Podatkovni model `Stranka` predstavlja uporabnika spletne banke, ki koristi bančne storitve. Za vsako stranko hranimo njen EMŠO, ime in priimek ter naslov stalnega prebivališča. Za podporo homomorfnim operacijam za vsako stranko pomnimo tudi pot do datoteke z njenim javnim ključem. Brez javnega ključa namreč spletna



Slika 5.1: Logični model podatkovne baze spletne banke. Podčrtani atributi so primarni ključi, atributi, predznačeni z znakom #, so tuji ključi, modro osenčena polja pa predstavljajo atribute, ki smo jih dodali k entitetnim tipom za podporo PHE, in so namenjeni shranjevanju poti do kriptografskih datotek.

banka ne more spreminjati zneska na transakcijskem računu stranke, zato ga je potrebno predhodno prenesti na spletni strežnik.

Stranka ima pri banki lahko odprt kvečjemu en transakcijski račun.

Transakcijski račun

S transakcijskim računom v podatkovni bazi spletne aplikacije hranimo podatek o trenutnem stanju na računu dane stranke. `Transakcijski račun` enolično identificira številka transakcijskega računa.

Stanje na transakcijskem računu stranke obravnavamo kot občutljiv podatek, zato shranjujemo njegov tajnopis. Tajnopis je shranjen v datoteki, v podatkovni bazi pravzaprav pomnimo zgolj pot do le-te na datotečnem sistemu strežnika. Zaradi optimizacije je omenjeni tajnopis v končni fazi vseboval podatke ne zgolj o trenutnem stanju na transakcijskem računu stranke, ampak tudi zgodovino vseh njegovih preteklih stanj.

Zgolj za potrebe sprotnega preverjanja pravilnosti procesiranja tajnopisov tekom razvoja in zaradi kasnejših meritev učinkovitosti strežnik shranjuje tudi čistopise stanj na transakcijskih računih.

Transakcija

S transakcijo zabeležimo prenos danega denarnega zneska z enega transakcijskega računa na drugega. V podatkovni bazi za transakcijo shranjujemo podatek o času njene izvedbe, namenu, znesku, ki se prenaša, ter številki transakcijskih računov obeh strank, udeleženih v transakciji (v nadaljevanju ju poimenujemo tudi *stranka na začetni* in *stranka na prejemni strani transakcije*).

Znesek za prenos med transakcijskima računoma želimo skriti pred spletno aplikacijo in zato shranjujemo njegov tajnopis. Ker pa mora v okviru homomorfne transakcije strežnik ta znesek odšteti z računa stranke na začetni strani transakcije in ga prišteti na račun stranke na prejemni strani, pravzaprav potrebujemo dva tajnopisa. Isti znesek je namreč potrebno enkrat zakriptirati z javnim ključem začetnika, drugič pa z javnim ključem prejemnika. Homomorfne transakcije bolj podrobno opišemo v razdelku 5.2.3.

Tajnopisa sta zopet shranjena v datotekah na strežniku. Podobno kot pri čistopisu stanja na transakcijskem računu pa smo za sprotno preverjanje pravilnosti s transakcijo shranjevali tudi čistopis zneska za prenos med računoma.

Zaradi omejitev uporabljene programske knjižnice za PHE so lahko stanja na transakcijskih računih strank in zneski, ki nastopajo v transakcijah, samo cela števila.

5.1.3 Aplikacijska logika

Že v prejšnjem razdelku z opisom podatkovnega modela spletne aplikacije smo lahko opazili prisotnost podatkovnih polj, namenjenih shranjevanju poti do datotek s kriptografskim materialom, ki je nujen za podporo polni homomorfni enkripciji. Podobno se tudi poslovna logika naše spletne banke na več mestih prepleta z logiko za podporo PHE. V nadaljevanju podajamo zgolj opis splošne in poslovne aplikacijske logike spletne aplikacije brez podrobnosti o funkcionalnostih nizkonivojske programske kode za podporo PHE.

Registracija in prijava

Spletna aplikacija omogoča registracijo novih uporabnikov in preprosto avtentikacijo registriranih uporabnikov s pomočjo uporabniškega imena in gesla. S postopkom registracije pridobimo večino podatkov, ki jih tekom življenjskega cikla

spletne aplikacije vzdržujemo o strankah.

Pregled tekočega stanja in drugih podatkov o transakcijskem računu

Po uspešni registraciji in prijavi v spletno aplikacijo z uporabniškim imenom in geslom lahko uporabnik pregleda podatke o svojem transakcijskem računu. Spletna banka predpostavlja predhodni obstoj transakcijskega računa in z njim povezanih podatkov v sistemu banke. Povedano drugače, novega transakcijskega računa ni mogoče odpreti preko spletne aplikacije.

Izvajanje bančnih transakcij

Prijavljen uporabnik z obstoječim transakcijskim računom lahko izvede novo transakcijo v obliki prenosa izbranega denarnega zneska na drug transakcijski račun, pri čemer zaradi enostavnosti predpostavljamo, da drug račun prav tako pripada uporabniku, ki je stranka naše spletne banke.

Pregled zgodovine transakcij

Prijavljen uporabnik, ki ima pri banki odprt transakcijski račun, lahko prav tako pregleduje zgodovino z njim povezanih transakcij.

5.2 Programska podpora PHE

V nadaljevanju predstavimo nizkonivojske programe za podporo polni homomorfni enkripciji, ki se uporabljajo na strani strežnika in odjemalca. Ti programi slonijo na uporabi programske knjižnice HElib [22], zato si najprej oglejmo nekaj lastnosti knjižnice.

5.2.1 Knjižnica HElib

Kljub dejstvu, da je polna homomorfna enkripcija relativno novo raziskovalno področje, je v trenutku pisanja tega besedila na voljo več odprtokodnih programskih knjižnic, ki implementirajo različne polne homomorfne enkripcijske sheme. Med njimi so tudi Scarab [37], FHEW [11] in HElib, ki so vse nizkonivojske knjižnice v programskem jeziku C ali objektno usmerjenem C++.

Za implementacijo polne homomorfne enkripcije s pomočjo knjižnice HElib smo se odločili, ker je slednja med vsemi trenutno najbolj napredna, vzdrževana in dobro dokumentirana, obenem pa tudi edina nudi eksperimentalno podporo večnitnemu izvajanju.

Osnovni koncepti HElib

Knjižnica HElib je napisana v programskem jeziku C++ in sloni na dveh matematičnih knjižnicah - GMP (*GNU Multiple Precision Arithmetic Library* [13]) in NTL (*Number Theory Library* [32]). HElib implementira nekoliko homomorfno enkripcijsko shemo Brakerski-Gentry-Vaikuntanathan (BGV) in za optimizacijo kompleksnosti osveževanja tajnopisov v postopku samozagona uporablja algoritem RLWE (angl. *Ring Learning with Errors*). Za pohitritev samozagona HElib omogoča tudi podporo večnitnemu izvajanju. Večnitno izvajanje lahko izkoristimo tudi za pohitritev nekaterih programskih rutin, ki slonijo na matematični knjižnici NTL [24].

Za lažje razumevanje opisa naših implementacij, ki bo sledil v razdelku 5.2.3, v naslednjih razdelkih podajamo opise ključnih konceptov in programskih struktur, s katerimi se srečamo pri razvoju programov s knjižnico HElib (povzeto po delih [21] in [22]).

Kriptografski kontekst Za inicializacijo ustreznih programskih struktur moramo v programih, ki uporabljajo HElib, določiti vrednosti nekaterih ključnih parametrov. Izbira vrednosti teh parametrov močno vpliva na učinkovitost programov v HElib. Vsi ti parametri so celoštevilski, za naše implementacije pa so najpomembnejši naslednji:

- parametra p in r, ki definirata velikost prostora čistopisov $\mathbb{F}_{p^r}$,
- parameter k, ki predstavlja število bitov v tajnopisih, namenjenih zagotavljanju varnosti,
- parameter L, ki določa kapaciteto šuma v tajnopisih. Njegova vrednost naj bi bila sorazmerna številu homomorfnih množenj, ki jih nameravamo v programu izvršiti nad posameznim tajnopisom.

Zaradi lažjega dostopa programskih objektov knjižnice do zgornjih parametrov HElib slednje shranjuje v posebno podatkovno strukturo - kriptografski kontekst. Vzpostavitev kriptografskega konteksta je prva stvar, ki jo moramo narediti v vsakem programu, ki uporablja HElib, ne glede na to, kakšno logiko program implementira.

Kriptografski ključi Pri uporabi HElib se srečamo z javnimi in zasebnimi kriptografskimi ključi, saj je enkripcijska shema BGV asimetrična. Tako javni (enkripcijski) kot tudi zasebni (dekripcijski) ključi so definirani relativno glede na predhodno definiran kriptografski kontekst. To pomeni, da sta javni in zasebni ključ vezana na konkretne kriptografske parametre, zato enkripcija in dekripcija v primeru uporabe napačnega kriptografskega konteksta ne delujeta pravilno.

Čistopisi Pred enkripcijo moramo podatke pretvoriti v obliko, ki jo pričakuje HElib. V HElib so čistopisi predstavljeni z vektorji. Na prostore čistopisov, ki jih lahko predstavimo v HElib, lahko gledamo kot na omejene vektorske prostore. Vektorji so namreč omejenih dolžin (od več sto do več tisoč elementov), prav tako pa je omejena tudi zaloga vrednosti elementov. Na velikost vektorjev in zalogo vrednosti njihovih elementov vpliva izbira kriptografskih parametrov p in r, vendar pa HElib trenutno podpira samo bitne in celoštevilske vektorje. Zalogo celoštevilskih vrednosti predstavlja potenca p^r (npr. za aritmetiko z nepredznačenimi 16-bitnimi celimi števili zadošča konfiguracija $p = 2$ in $r = 16$).

Primera dveh čistopisov, pretvorjenih v obliko, kot jo pričakuje HElib, prikazuje slika 5.2.

Tajnopisi Tajnopisi so operandi in rezultati homomorfnih operacij. Zaradi vektorske predstavitve čistopisov so tudi tajnopisi v HElib predstavljeni z vektorji. To omogoča t. i. *pakiranje tajnopisov* (angl. *ciphertext packing*), ki je zaradi podpore vektorskemu homomorfnemu procesiranju ena izmed pomembnih optimizacij knjižnice. Primer tajnopisa, ki je nastal kot rezultat enkripcije v HElib, lahko vidimo na sliki 5.3.

HElib podpira seštevanje, odštevanje in množenje tajnopisov, poleg tega pa še prištevanje in odštevanje konstante tajnopisu, množenje tajnopisa s konstanto,

```
[100 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 ...]
```

(a) Primer predstavitve skalarja pred enkripcijo. Uporabimo samo en element vektorja.

```
[10 80 50 55 85 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 ...]
```

(b) Primer predstavitve vektorja podatkov pred enkripcijo. Uporabimo toliko elementov, kot jih potrebujemo.

Slika 5.2: Predstavitev skalarja (a) in vektorja (b) v obliki čistopisa, primernega za uporabo v HElib. Primera predpostavljata, da so neizkoriščeni elementi vektorja ničelni, v splošnem pa lahko uporabimo vse elemente vektorja.

```
[1765627 417514 2170757 308468 387422 2289163 4256871 ...]
```

Slika 5.3: Primer predstavitve tajnopisa v programski knjižnici HElib.

negacijo, zamik in vrtenje posameznega tajnopisa. Zaradi predstavitve z vektorji so vse omenjene operacije vektorske.

Za uspešno izvedbo homomorfnih operacij mora entiteta, ki procesira tajnopise, poznati javni ključ, ki je bil uporabljen za ustvarjanje tajnopisa. Če ima homomorfna operacija več kot en kriptiran operand (npr. pri seštevanju, odštevanju in množenju tajnopisov), jo lahko uspešno izvedemo le v primeru, da sta oba tajnopisa zakriptirana z istim javnim ključem.

5.2.2 Obstoječi primeri uporabe PHE

Trenutno je uporaba PHE omejena na prototipne implementacije v raziskovalne namene. Vsi nam znani primeri uporabe iz literature polno homomorfno enkripcijo uporabljajo za najrazličnejše namene, od varnega procesiranja signalov do razpoznavanja obrazov. Vendar pa so omenjena dela osredotočena predvsem na pri-

lagoditev algoritmov za homomorfno procesiranje ali na izboljšanje učinkovitosti obstoječih. Posledično obstaja že kar nekaj (kot bomo videli v nadaljevanju v primerjavi z našim primerom uporabe PHE s pomočjo HElib v okviru spletne aplikacije) relativno kompleksnih primerov uporabe polne homomorfne enrkipcije.

Pomanjkljivosti obstoječih primerov uporabe

Pri veliki večini omenjenih primerov uporabe nas je zmotilo dejstvo, da ne upoštevajo ločitve programske kode na strežniški in odjemalčev del. Na primer, iz del [30] in [34] je razvidno, da se vsa programska koda izvaja na strežniku, oziroma da njihove implementacije ne obravnavajo vloge odjemalca kot vira podatkov za nadaljnje strežniško procesiranje. Omenjene implementacije s tem zaobidejo režijsko delo, ki nastane kot posledica upravljanja s ključi in ustvarjanja tajnopisov na strani odjemalca.

Povedano drugače, vzpostavitev kriptografskega konteksta, izpeljava javnega in zasebnega ključa, enkripcija, homomorfno procesiranje in dekripcija se v omenjenih primerih uporabe v celoti izvajajo na strani strežnika. Za kakršno koli realno aplikacijo PHE v oblaku je to nesprejemljivo, saj strežnik ne sme imeti vpogleda v zasebne ključe odjemalcev. Edina naloga, ki naj bi jo imel strežnik, je homomorfno procesiranje in z njim povezana predhodna obnovitev kriptografskega konteksta in javnega ključa odjemalca. Podpora slepemu strežniškemu procesiranju (tj. procesiranju, kjer strežnik nima vpogleda v pravi pomen podatkov, ki jih procesira) občutljivih podatkov v javnem oblaku je nenazadnje ena izmed poglavitnih stvari, zaradi katerih je homomorfna enkripcija v zadnjih letih tako aktualna.

Razkorak med obstoječimi rešitvami, ki slonijo na uporabi PHE, in rešitvami, kakršne naj bi bile v realnih okoljih računalniških oblakov, nas je spodbudil k razvoju programja za podporo PHE v oblaku.

5.2.3 Opis implementacije v C++

Logična ločitev na del operacij, ki se morajo izvršiti na strani odjemalca, in tiste, ki se lahko izvršijo na strani strežnika, je v primeru uporabe polne homomorfne enkripcije očitna. Na aplikativnem nivoju polne homomorfne enkripcije zato lahko na nek način govorimo o modelu odjemalec-strežnik.

Strežnik - HE_Server

Naša aplikacija spletne banke mora za operacije, ki zahtevajo ustvarjanje ali posodabljanje tajnopisov stanj na transakcijskih računih odjemalcev, klicati funkcije programa v C++. Slednji za implementacijo podpore PHE uporablja knjižnico HElib. Ko spletni strežnik od končnega uporabnika preko spletne aplikacije dobi vse za izvedbo dane operacije potrebne vhodne podatke, jih mora predati strežniškemu programu v C++. Program v jeziku C++, ki se izvaja na strežniku v računalniškem oblaku, smo poimenovali `HE_Server`. Preden se posvetimo funkcionalnostim programa `HE_Server`, si najprej oglejmo, kako spletna aplikacija kliče njegove funkcije.

Klici funkcij programa HE_Server iz spletne aplikacije S pomočjo knjižnice za interoperabilnost med programskima jezikoma Python in C++ *Boost.Python* [6] smo program `HE_Server` zgradili kot deljeni objekt (angl. *shared object*). Na ta način smo omogočili uvoz funkcij programa C++ v obliki modula v Pythonu in iz kode spletne aplikacije v Pythonu neposredno klicali njegove funkcije.

Denimo, da smo v programu `HE_Server` predhodno definirali funkciji `init_ciphertexts` in `transaction` z želenimi argumenti. Da ju lahko v nadaljevanju kličemo iz kode v programskem jeziku Python, moramo na konec programa `HE_Server` dodati naslednje definicije:

```
BOOST_PYTHON_MODULE(HE_Server)
{
    using namespace boost::python;
    def("transaction", transaction);
    def("initCiphertexts", init_ciphertexts);
},
```

nato pa lahko izbrano funkcijo iz spletne aplikacije pokličemo na način, ki ga prikazuje spodnji primer:

```
import HE_Server
result = HE_Server.transaction(args),
```

kjer `args` predstavlja dejanske argumente funkcije. Za ustrezno pretvorbo podatkovnih tipov argumentov in rezultatov funkcij med programskima jezikoma Python in C++ poskrbi knjižnica *Boost.Python*.

Funkcije programa HE_Server Kot smo videli v zgornjem primeru, ima program `HE_Server` dve funkciji, in sicer:

1.) funkcijo za inicializacijo tajnopisa s stanjem na transakcijskem računu uporabnika. To funkcijo spletna aplikacija pokliče, ko končni uporabnik posreduje svoj javni ključ. Njena naloga je ustvarjanje nove datoteke s tajnopisom začetnega stanja na transakcijskem računu danega uporabnika, pri čemer uporabnikom nastavimo poljuben začetni znesek. To naredimo zgolj zaradi enostavnosti, da omenjene operacije ni potrebno izvesti na odjemalcu, bi bilo pa bolj smiselno, če je ne bi izvajali na strani strežnika.

2.) Funkcijo za izvedbo homomorfnih bančnih transakcij. Rezultat posamezne homomorfne transakcije sta novi stanji na transakcijskih računih obeh udeleženih odjemalcev, zato bo ta funkcija s pomočjo homomorfnega procesiranja pravzaprav posodabljala tajnopise s stanji na transakcijskih računih uporabnikov spletne banke. Homomorfne bančne transakcije natančneje opišemo v naslednjem razdelku.

Za izvedbo vseh ostalih s PHE povezanih operacij, ki bi jih končni uporabnik spletne aplikacije utegnil zahtevati od strežnika (na primer branje trenutnega stanja na računu), poskrbi spletna aplikacija. Slednja namreč hrani tajnopise s stanji na računih vseh odjemalcev in njihove javne ključe, ki jih mora na zahtevo zgolj posredovati odjemalcu, zaradi česar ni potreben zagon funkcij programa `HE_Server`.

Homomorfne transakcije Naj bo m znesek, ki ga želi prvi odjemalec (odjemalec A) prenesti na transakcijski račun drugega odjemalca (odjemalca B). Bančna transakcija potem obsega dve operaciji, in sicer odštevanje zneska m z računa odjemalca A in prištevanje istega zneska na račun odjemalca B. V naši spletni banki se obe operaciji transakcije izvedeta nad tajnopisi s pomočjo polne homomorfne enkripcije.

Dejstvo, da transakcija učinkuje na računa dveh odjemalcev, še dodatno obremeni odjemalca A, torej odjemalca, ki začenja transakcijo. Strežnik spletne banke

namreč ne more uporabiti tajnopisa zneska, ki je bil zakriptiran z javnim ključem odjemalca A ($E_{K_{j_A}}(m)$, kjer je K_{j_A} javni ključ odjemalca A), za izvedbo homomorfne operacije nad stanjem na računu odjemalca B. Prav tako pa sama ne more dekriptirati tajnopisa zneska, ki ga dobi od odjemalca A (saj ne pozna njegovega zasebnega ključa K_{z_A}), zato je nujno, da za ustvarjanje tajnopisa zneska, zakriptiranim z javnim ključem odjemalca B ($E_{K_{j_B}}(m)$), poskrbi odjemalec A. Pri tem je potrebno omeniti, da bi bilo nesmotrno zahtevati neposredno sodelovanje odjemalca B, ki je kot tipičen odjemalec oblačnih storitev večino časa nedosegljiv.

Za uspešno izvedbo transakcije mora torej odjemalec A od strežnika najprej pridobiti javni ključ odjemalca B, K_{j_B}. Odjemalec A namreč pozna številko transakcijskega računa odjemalca B, ne pa tudi njegovega javnega ključa. Po drugi strani pa mora spletna banka zaradi podpore homomorfnemu procesiranju hraniti javne ključe vseh svojih odjemalcev. Na podlagi številke transakcijskega računa, ki pripada danemu odjemalcu, zato spletna banka lahko poišče pripadajoč javni ključ in ga v obliki datoteke posreduje odjemalcu, ki začenja transakcijo.

Odjemalec A nato zakriptira čistopis zneska, m, enkrat s svojim javnim ključem (rezultat je tajnopis $E_{K_{j_A}}(m)$), enkrat pa z javnim ključem prejemnika (rezultat je tajnopis $E_{K_{j_B}}(m)$). Za uspešno izvedbo homomorfnih različic obeh operacij transakcije mora torej odjemalec A spletni banki posredovati dva tajnopisa.

Spletna banka lahko zatem izvede bančno transakcijo kot par dveh homomorfnih operacij, ki vključujeta izračun tajnopisov novih stanj na računih udeleženih odjemalcev:

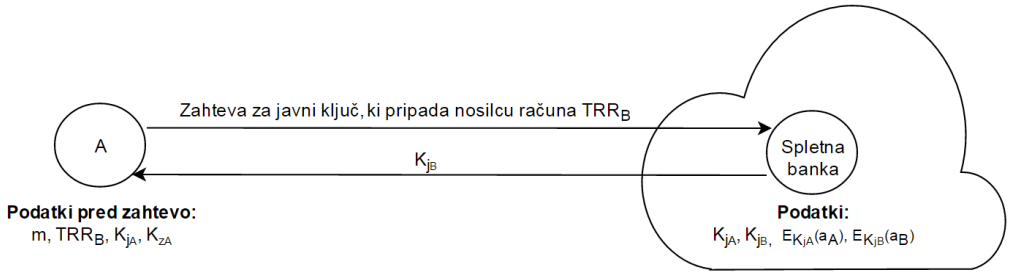
$$E_{K_{j_A}}(a'_A) = E_{K_{j_A}}(a_A) - E_{K_{j_A}}(m), \tag{5.1}$$

$$E_{K_{j_B}}(a'_B) = E_{K_{j_B}}(a_B) + E_{K_{j_B}}(m), \tag{5.2}$$

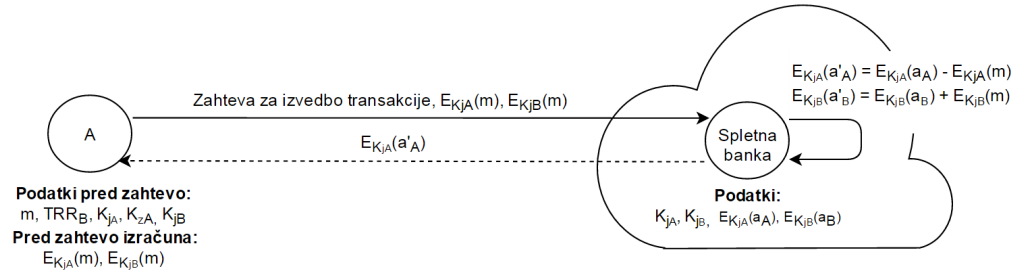
kjer a_A in a_B označujeta čistopisa stanj na računih odjemalcev A in B pred transakcijo, a'_A in a'_B pa čistopisa stanj odjemalcev A in B po uspešno izvedeni transakciji.

Izmenjavo podatkov med odjemalcem in spletno banko pred homomorfno transakcijo in po njej grafično prikazujeta sliki 5.4 in 5.5.

Za podporo homomorfnim transakcijam v skladu z zgornjim opisom moramo na implementacijskem nivoju (tj. na nivoju funkcije za izvedbo homomorfne bančne



Slika 5.4: Komunikacija med odjemalcem in spletno banko z namenom pridobivanja javnega ključa odjemalca, ki ima vlogo prejemnika denarnega zneska.



Slika 5.5: Komunikacija med odjemalcem in spletno banko tik pred homomorfno transakcijo in tik po njej. S prekinjeno črto smo označili strežnikov odgovor, ki vključuje tajnopis novega stanja na računu začetnika transakcije - ta korak seveda ni nujen, vendar smo ga v naši spletni banki kljub temu implementirali.

transakcije v programu `HE_Server`) izvesti naslednje naporedje korakov:

1.) vzpostaviti kriptografski kontekst,

2.) iz datotek prebrati in obnoviti (*deserializirati*) javna ključa obeh udeležencev transakcije,

3.) iz datotek prebrati in deserializirati tajnopisa s stanji na transakcijskih računih obeh udeležencev transakcije,

4.) iz datotek prebrati in deserializirati tajnopisa zneska, ki se prenaša med transakcijskima računoma, zakriptiranima z javnima ključema obeh udeležencev transakcije,

5.) izvesti homomorfno transakcijo, v okviru katere začetniku transakcije odštejemo, prejemniku pa prištejemo kriptiran znesek,

6.) serializirati in v datoteki zapisati nova tajnopisa stanj na računih obeh udeležencev transakcije.

Beleženje zgodovine s homomorfnimi transakcijami Opis homomorfnih transakcij iz prejšnjega razdelka predvideva, da strežniški program `HE_Server` za posamezno transakcijo izvede samo dve homomorfni operaciji (v skladu z enačbama 5.1 in 5.2), in sicer po eno za vsako od strank. Na ta način smo v začetni različici programa `HE_Server` posodabljali stanje na računu strank, pri čemer smo staro stanje na računu (tajnopis pred izvedbo transakcije) vedno prepisali z novim. V tem primeru smo s tajnopisom vedno predstavili eno samo število - trenutno stanje na transakcijskem računu.

Ker pa smo v spletni aplikaciji želeli podpreti pregled zgodovine bančnih transakcij, opisan pristop ni več zadoščal. Najbolj intuitivna rešitev bi bila, da tajnopisa s stanjem pred izvedbo transakcije ne bi vsakič prepisovali z novim, ampak bi novega shranili v ločeno datoteko. V tem primeru bi število datotek s tajnopisi naraščalo sorazmerno s številom transakcij, kar s stališča strežnika v računalniškem oblaku praviloma ne bi smelo biti problematično zaradi velike količine razpoložljivega prostora. Vendar pa bi obenem moral tudi odjemalec ob vsaki zahtevi za pregled zgodovine transakcij - ob predpostavki da tajnopisov ne predpomnimo - iz strežnika banke prenesti in dekriptirati vse tajnopise ali pa vsaj zadnjih nekaj.

Prenos večje količine podatkov iz strežnika spletne banke in povečan strošek dekripcije bi se v tem primeru odražal v daljšem nalaganju spletne strani za pregled zgodovine transakcij. Namesto implementacije predpomnenja, ki bi ga bilo potrebno realizirati na strani odjemalca, smo razvili boljšo rešitev, ki sicer bolj obremeni strežnik, vendar pa od odjemalca še vedno zahteva prenos in dekripcijo enega samega tajnopisa. Ta optimizacija temelji na uporabi pakiranja tajnopisov, ki ga ponuja HElib.

Spomnimo se, da so pred enkripcijo čistopisi predstavljeni z vektorji, prav tako pa so vektorji tudi tajnopisi. Naša prvotna rešitev je za pomnenje stanja na računu uporabljala samo prvi element vektorja, ostali pa so bili ničelni. Za

pomnenje zgodovine transakcij, ki se odraža v spremembah na stanju transakcijskega računa stranke, pa program `HE_Server` sproti povečuje število neničelnih elementov vektorja. Zadnji neničelni element tako predstavlja trenutno stanje na računu odjemalca, elementi pred njim pa kronološko urejeno zaporedje prejšnjih stanj. Za izračun novega stanja program `HE_Server` poleg tajnopisa trenutnega stanja (ki je v tem primeru pravzaprav tajnopis s trenutnim in prejšnjimi stanji) in tajnopisa zneska za prenos potrebuje še zaporedno številko transakcije glede na transakcijski račun ter vlogo odjemalca pri transakciji. Z uporabo homomorfnih seštevanj, odštevanj in premikov program `HE_Server` izračuna novo stanje v skladu z algoritmom 1.

Algoritem 1: Posodabljanje stanja na transakcijskem računu (optimizacija s pakiranjem tajnopisov)

Vhodi: Tajnopis z zneskom za prenos t, tajnopis s trenutnim in prejšnjimi stanji a, zaporedna številka transakcije i, indikator ali tajnopis pripada odjemalcu na začetni strani transakcije $zacel\_transakcijo$

Izhod : Tajnopis a' z novim stanjem na transakcijskem računu

$a' \leftarrow a$;
$t = he\_shift(t, i)$;
$a' = he\_shift(a', -(i-1))$;
$a' = he\_shift(a', i)$;
if $zacel\_transakcijo$ **then**
    $a' = he\_subtract(a', t)$;
else
    $a' = he\_add(a', t)$;
end
$a' = he\_add(a, a')$;
return a';

Algoritem 1 ne obravnava robnega primera, ko zaporedna številka transakcije doseže velikost vektorja - velikosti vektorjev so namreč fiksne in omejene. V tem primeru bi morali implementirati še nekaj dodatne logike, ki bi ustvarila nov taj-

nopis in vanj zabeležila novo stanje. Ta bi se izvedla razmeroma redko, saj imajo vektorji od nekaj sto do nekaj tisoč elementov, odvisno od uporabljenih kriptografskih parametrov. Za prikaz celotne zgodovine stanj na transakcijskem računu pa bi v tem primeru odjemalec še vedno moral k sebi prenesti in dekriptirati za več velikostnih redov manj tajnopisov, kot če te optimizacije ne bi implementirali.

Cena enostavnejše in hitrejše dekripcije zgodovine stanj na transakcijskih računih strank pa so dražje homomorfne transakcije, saj izvedba transakcije zahteva deset homomorfnih operacij (po pet za vsakega odjemalca) namesto prvotnih dveh.

Odjemalec

Za razliko od strežniškega programa za podporo PHE, ki omogoča zgolj inicializacijo tajnopisa z začetnim stanjem na transakcijskem računu in izvajanje homomorfnih bančnih transakcij, program v C++ na strani odjemalca vsebuje funkcije za:

- **enkripcijo** celoštevilskega operanda, ki v nadaljevanju nastopa kot znesek, ki se v okviru transakcije prenaša med računoma odjemalcev. Slednja najprej vzpostavi kriptografski kontekst, nato pa iz datoteke prebere in deserializira javna ključa obeh uporabnikov, katerih računa sta udeležena v transakciji. Za tem dvakrat izvede enkripcijo danega zneska, po enkrat z vsakim od obeh javnih ključev. Kot rezultat vrne ustvarjena tajnopisa.

- **Dekripcijo** tajnopisa s stanjem oz. z zgodovino stanj na transakcijskem računu. Ta funkcija po vzpostavitvi kriptografskega konteksta deserializira zasebni ključ uporabnika, ki poizveduje po znesku na svojem transakcijskem računu. S pomočjo zasebnega ključa dekriptira tajnopis in kot rezultat vrne čistopis odjemalčevih podatkov.

- **Inicializacijo**, ki obsega generiranje kriptografskih ključev (zasebnega in javnega). Funkcija oba ključa shrani v datoteko. Ker ne želimo, da ima strežnik spletne banke vplogled v zasebne ključe odjemalcev, smo to funkcijo implementirali na strani odjemalca.

Izdelali smo dva podobna programa v jeziku C++ z implementacijami zgornjih funkcij, ki smo ju pomenovali `HE_Client` in `HE_RemoteClient`. Prvega uporabljamo v okviru pristopa z razširitvijo za brskalnik Chrome (podrobneje opisanim

v razdelku 5.4), drugega pa v okviru pristopa z dodatnim posvečenim strežnikom (podrobneje opisanim v razdelku 5.5). Programa se razlikujeta zgolj v načinu pridobivanja parametrov, potrebnih za izvedbo izbrane operacije.

Serializacija in deserializacija

Kriptografski material, predvsem javne ključe in tajnopise, je v naših rešitvah potrebno pogosto prenašati preko omrežja. S kriptografskimi ključi in tajnopisi rokujemo na nivoju programov C++, v katerih so slednji predstavljeni kot programski objekti. Ker pa želimo iste kriptografske ključe uporabiti večkrat, programi, znotraj katerih slednji obstajajo kot objekti pa se relativno hitro zaključijo, potrebujemo način, kako jih shraniti in ob ponovnem zagonu programov C++ obnoviti. Mehanizma za pretvorbo objektov v obliko, primerno za trajno shranjevanje ali pošiljanje preko omrežja, in njihovo ponovno obnovitev (t. i. *rekonstrukcijo*) znotraj programa imenujemo *serializacija* in *deserializacija* [7].

V naših programih po serializaciji v sklopu različnih operacij uporabljamo obe različici - včasih shranjevanje v datoteko, včasih pa zgolj pošiljanje serializiranega objekta preko omrežja. Katero različico uporabljamo, je odvisno od tega, ali nameravamo serializirani objekt obnoviti enkrat ali večkrat. Odločili smo se, da objekte, ki jih nameravamo obnoviti le enkrat, po serializaciji zgolj pretočimo na stran prejemnika. Prejemnik nato objekt deserializira, ga uporabi za neko operacijo, nato pa zavrže. S tem se izognemo nepotrebnemu pisanju v datoteko na eni strani in branju datoteke na drugi strani. Konkretno to naredimo z objekti, ki predstavljajo tajnopise operandov, ki nastopajo kot zneski za prenos med transakcijskima računoma v sklopu transakcije.

V nasprotnem primeru pa kriptografske ključe in tajnopise s stanji na transakcijskih računih po serializaciji shranjujemo v datoteke, saj jih znotraj življenjskega cikla naše aplikacije večkrat potrebujemo in jih zato ne želimo zavreči. V primeru da jih želimo prenesti z ene naprave na drugo (npr. javni ključ z naprave odjemalca na strežnik), torej pošiljamo datoteke.

Večnitno izvajanje

Preizkusili smo tudi večnitno izvajanje na nivoju nizkonivojskih programskih rutin HElib, ki temeljijo na podporni programski knjižnici NTL. V ta namen smo morali

z drugačnimi parametri ponovno prevesti in zgraditi omenjeni knjižnici in vse naše programe v C++ (`HE_Server, HE_Client` in `HE_RemoteClient`). V ukaze datotek za izgradnjo naših programov v C++ smo dodali stikali za podporo večnitnemu izvajanju (`-DFHE_DCRT_THREADS` in `-pthread`). Ob prisotnosti omenjenih stikal naj bi HElib samodejno poskrbela za večnitno izvajanje.

5.3 Pristop s prenosom knjižnice HElib v spletni brskalnik

Naš prvi poskus realizacije programske rešitve, ki bi odjemalcu omogočala uporabo funkcionalnosti knjižnice HElib, je bil prenos (angl. *porting*) omenjene knjižnice za neposredno uporabo v spletnem brskalniku. Slednje bi odjemalcu omogočalo njeno uporabo iz skript v programskem jeziku JavaScript, ki praviloma sestavljajo čelni del (angl. *front end*) vsake spletne aplikacije. Na ta način bi na strani odjemalca potrebovali samo spletni brskalnik; tako bi se izognili potrebi po uvedbi dodatnih komponent v arhitekturo odjemalca, na primer razširitvi za spletni brskalnik ali dodatnemu posvečenemu strežniku, opisanima v razdelkih 5.4 in 5.5.

Opisan pristop s prenosom funkcionalnosti programske knjižnice HElib v spletni brskalnik se nam je zdel intuitiven, zato smo se ga lotili najprej. Kljub temu pa se je že sam postopek prenosa knjižnice izkazal za tehnično zahtevno in dolgotrajno opravilo, ki nam ga ni uspelo realizirati.

5.3.1 Postopek prenosa

Prenosa smo se lotili na dva načina, oziroma z uporabo dveh različnih odprtokodnih orodij, ki podpirata zagon lastnih (angl. *native*) aplikacij v spletnem brskalniku in njihovo predelavo v ustrezno obliko, in sicer *NaCl* (t. i. *Native Client*) ter *Emscripten*.

NaCl

NaCl je tehnologija, ki so jo pri podjetju Google razvili za optimizacijo predvsem računsko intenzivnih spletnih aplikacij, ki tečejo v brskalniku Google Chrome, na varen način. Omogoča jim namreč zagon kode v programskih jezikih C in C++

neposredno iz spletnega brskalnika, zaradi česar se lahko izvajajo veliko hitreje, kot če bi ekvivalentne operacije izvajali v programskem jeziku JavaScript [44]. Prav tako pa spletnim aplikacijam omogoča ponovno uporabo obstoječih programskih knjižnic v jezikih C in C++, kar je bil tudi glavni razlog, da smo želeli uporabiti NaCl.

Ker niti HElib niti ustrezna verzija nobene od njenih podpornih programskih knjižnic še ni bila preoblikovana v obliko, primerno za zagon iz brskalnika, smo se tega postopka lotili sami. Izkazalo se je, da prenos kode omenjenih knjižnic v spletni brskalnik presega zastavljeni obseg pričujočega dela, zato ga nismo realizirali.

Emscripten

Emscripten je orodje, pisano v programskem jeziku JavaScript, ki omogoča prevajanje zbirne kode LLVM (angl. *Low Level Virtual Machine*) v JavaScript [45]. Knjižnico HElib in vse knjižnice v jeziku C++, ki jih HElib uporablja, smo namesto v izvršljive binarne datoteke prevedli v kodo LLVM, slednjo pa s pomočjo orodja Emscripten še v jezik JavaScript. Kot rezultat smo dobili ogromno datoteko JavaScript z okrog 150 tisoč vrsticami programske kode, vendar funkcij knjižnice HElib nismo mogli klicati, ne da bi pri tem naleteli na napačno delovanje.

5.3.2 Težave pri postopku prenosa

Pri uporabi obeh orodij smo naleteli na podobne težave, ki so v obeh primerih izvirale iz dejstva, da HElib sloni na uporabi dveh matematičnih knjižnic v jeziku C++, kjer je ena odvisna od druge. Kaskada treh programskih knjižnic, kjer ima poleg tega še vsaka povsem drugačen sistem za izgradnjo, se je izkazala za problematično. Spreminjati smo morali namreč datoteke za konfiguracijo in izgradnjo (t. i. *Makefile* datoteke) vsake izmed omenjenih treh knjižnic, vendar nam le-teh ni uspelo ustrezno prilagoditi, tako da bi se knjižnica HElib uspešno zgradila.

Ker nam ob uporabi dveh različnih tehnologij za prenos programskih knjižnic C++ v spletni brskalnik in dolgotrajnem razhroščevanju ni uspelo odpraviti vseh težav, smo ta pristop opustili in poiskali alternativne možnosti. Predlagali smo še dva alternativna pristopa, predstavljena v naslednjih razdelkih, ki smo ju uspešno realizirali.

5.4 Pristop z razširitvijo za spletni brskalnik Chrome

Razširitev spletnega brskalnika je vtičnik (angl. *plug-in*), ki razširja njegove funkcionalnosti. Razširitve pogosto uporabljamo za spreminjanje uporabniškega vmesnika brskalnika ali spletnih strani in so posledično običajno realizirane s pomočjo spletnih tehnologij na čelni strani kot so HTML, CSS in JavaScript [18].

Spletni brskalniki svojim razširitvam v splošnem nudijo podporo za komunikacijo z lastnimi aplikacijami računalniškega sistema. V lastno aplikacijo, ki je v našem primeru program v C++ `HE_Client`, je v tem primeru potrebno vgraditi podporo omenjeni komunikaciji. Trenutno ne obstaja splošna rešitev za komunikacijo med spletnim brskalnikom (oz. njegovo razširitvijo) in lastnimi aplikacijami, ki bi bila kompatibilna z večimi brskalniki, zato smo se zaradi njegove razširjenosti osredotočili za implementacijo razširitve za brskalnik Chrome. Slednje seveda ne pomeni, da v ostalih spletnih brskalnikih komunikacije z lastnimi aplikacijami ni možno implementirati - ravno nasprotno, za spletne brskalnike Firefox, Safari, Internet Explorer in Opera obstajajo alternative, ki omogočajo natanko to.

5.4.1 Zgradba razširitve

Osnovno zgradbo razširitve za brskalnik Chrome podamo v njeni nastavitveni datoteki (t. i. *manifest*) v formatu JSON. V njej navedemo ime in opis razširitve, njene sestavne dele, njihove vrste in morebitna dovoljenja, ki jih razširitev potrebuje za pravilno delovanje.

Naša razširitev je brez uporabniškega vmesnika in v celoti temelji na tehnologiji JavaScript. Sestavljena je iz:

- **množice skript za dostop do vsebine spletnih strani in njihovo spreminjanje**. Te skripte lahko preverjajo stanje gradnikov spletnih strani naše aplikacije preko objektnega modela dokumentov (angl. *Document Object Model, DOM*) in jih po potrebi spreminjajo. Zaznajo lahko razne dogodke na spletni strani, ki nastanejo zaradi interakcije uporabnika z njenimi gradniki ali pa zaradi programskega spreminjanja le-teh, in se nanje odzovejo. Taka dogodka sta na primer klik na gumb ali sprememba vsebine besedilnega

polja.

Za vsako skripto iz te skupine moramo podati seznam vzorcev, ki se ujemajo z enotnimi naslovi virov (angl. *uniform resource locator, URL*) spletne aplikacije. Ko spletni brskalnik zazna ujemanje s predpisanim vzorcem, torej ko uporabnik obišče dani URL, brskalnik poskrbi za to, da se izvede pripadajoča skripta za spreminjanje vsebine [15].

- **Skripte, ki se izvaja v ozadju**. Ta skripta ne more dostopati do vsebine spletne strani, lahko pa zažene proces, v katerem se izvaja lasten program `HE_Client`, in z njim komunicira. Ta vrsta skript je v splošnem namenjena podpori dolgo trajajočim opravilom [16]. V našem primeru to opravilo ni funkcija v jeziku JavaScript, ampak lastni program `HE_Client`.

Naša razširitev od spletnega brskalnika Chrome torej potrebuje tri dovoljenja, in sicer:

- dovoljenje za dostop do vsebine aktivnega zavihka v brskalniku (potrebujemo ga za pravilno delovanje vseh skript, ki spreminjajo vsebino spletnih strani),
- dovoljenje za izvajanje v ozadju (potrebujemo ga za skripto, ki se izvaja v ozadju),
- dovoljenje za uporabo lastnega sporočanja (potrebujemo ga za komunikacijo med skripto, ki se izvaja v ozadju, in programom `HE_Client`).

5.4.2 Komunikacija med sestavnimi deli razširitve

Skripte za spreminjanje vsebine spletnih strani s skripto za izvajanje v ozadju komunicirajo s pomočjo pošiljanja sporočil. Obe vrsti skript lahko pošiljata sporočila drugi vrsti oz. poslušata za morebitnimi prihajajočimi sporočili. Sporočila so v formatu JSON, komunikacija pa je lahko sinhrona (blokira izvajanje pošiljatelja do prejema odgovora) ali asinhrona (izvajanje pošiljatelja se nadaljuje, ob prejemu sporočila se sproži povratni klic). Mi smo uporabljali asinhrono pošiljanje sporočil, da bi ohranili nemoteno delovanje skript za spreminjanje vsebine med čakanjem na rezultate, ki jih sporoči skripta za izvajanje v ozadju.

V kontekstu razširitve za brskalnik se komunikacija s pošiljanjem sporočil najpogosteje uporablja za vzpostavitev kratkoživega komunikacijskega kanala, v katerem se med skriptama običajno prenese le en par sporočilo-odgovor.

5.4.3 Komunikacija med razširitvijo in programom v C++

Nasprotno pa za vzpostavitev dolgoživih komunikacijskih kanalov, po katerih želimo prenesti več sporočil in odgovorov, razširitve za Chrome uporabljajo t. i. *vrata* (angl. *port*). Vanje pošiljamo in iz njih prejemamo sporočila, pri čemer vrata lahko povezujejo različne sestavne dele razširitev ali pa razširitev in lastni program. Slednja smo uporabili za prenašanje sporočil med skripto razširitve za izvajanje v ozadju in programom `HE_Server`. Tak način komunikacije imenujemo lastno sporočanje (angl. *native messaging*).

Za vzpostavitev lastnega sporočanja moramo v nastavitvenih datotekah brskalnika Chrome najprej registrirati gostitelja, ki predstavlja lastni program. V nastavitveni datoteki gostitelja moramo poleg poti do izvedljive datoteke lastnega programa podati tudi identifikator razširitve brskalnika, ki bo lahko z gostiteljem komunicirala. Brskalnik nato na zahtevo razširitve v ločenem procesu zažene gostitelja in vzpostavi vrata do procesa, preko katerega lahko z gostiteljem komunicira. Gostitelj lahko prebere sporočila iz razširitve preko standardnega vhoda, svoja sporočila pa razširitvi pošlje preko standardnega izhoda [17]. Komunikacija med razširitvijo in gostiteljem je torej lahko dvosmerna.

Protokol lastnega sporočanja natančno predpisuje strukturo sporočil v formatu JSON, ki se pretakajo med razširitvijo in gostiteljem. Komunikacija je s strani razširitve asinhrona - če razširitev od gostitelja pričakuje odgovor, je o prispelem odgovoru obveščena s povratnim klicem.

5.4.4 Opis delovnega toka

Glede na to, katero spletno stran aplikacije spletne banke končni uporabnik obišče v brskalniku, se najprej naloži ustrezna skripta razširitve, ki ima dostop do njene vsebine.

Prikaz stanja na računu in zgodovine bančnih transakcij

Na spletnih straneh za prikaz trenutnega stanja na transakcijskem računu in pregled zgodovine transakcij spletna aplikacija v odgovoru pošlje tajnopis z zgodovino stanj. Spletna banka v tem primeru ne pošlje datoteke s tajnopisom, ampak samo njeno vsebino. Ne glede na to, ali tajnopis uporabniku prikažemo ali ne, je njegova vsebina dostopna na spletni strani, zato ga lahko ustrezna skripta razširitve prebere in ga preda skripti, ki se izvaja v ozadju. Ta v ločenem procesu zažene program `HE_Client`, odpre vrata do procesa in mu preda tajnopis. Program `HE_Client` nato s pomočjo zasebnega ključa uporabnika, ki ga poišče na predhodno določeni lokaciji v datotečnem sistemu, dekriptira tajnopis in preko vrat skripti razširitve, ki se izvaja v ozadju, preda rezultat (čistopis). Slednja rezultat preda ustrezni skripti za spreminjanje vsebine in obdela rezultat - na primer, izlušči samo trenutno stanje na računu ali pa celotno zgodovino stanj. Nazadnje poskrbi še za ustrezno predstavitev rezultata in ga prikaže končnemu uporabniku na predvidenem mestu na spletni strani.

Priprava na homomorfno transakcijo

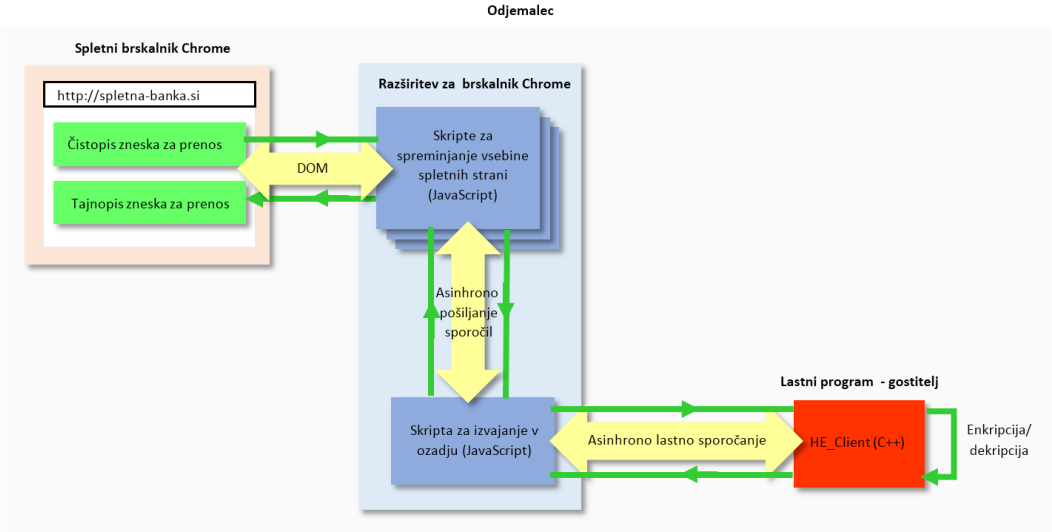
Če želi končni uporabnik preko spletne aplikacije sprožiti bančno transakcijo, je postopek podoben opisu iz prejšnjega razdelka, le da ustrezna skripta razširitve za dostop do vsebine spletne strani prebere čistopise uporabnikovih podatkov in ne tajnopisov, ki jih je poslal strežnik. Uporabnik mora namreč v besedilna polja vnesti znesek transakcije, namen in številko transakcijskega računa, na katerega želi prenesti denarni znesek. Ob kliku na gumb za začetek transakcije skripta prebere znesek za prenos, ga posreduje skripti, ki se izvaja v ozadju, ta pa vzpostavi vrata do procesa, v katerem se izvaja program `HE_Client`. Slednji preveri, ali je v datotečnem sistemu na predvideni lokaciji shranjen javni ključ uporabnika, kateremu želi končni uporabnik nakazati denar.

Če ključa prejemnika ni na predvideni lokaciji, se program `HE_Client` zaključi. V tem primeru lahko uporabnik javni ključ prejemnika zahteva od spletne banke, ga prenese k sebi in ga shrani na predvideno lokacijo.

Če pa v datotečnem sistemu na predvideni lokaciji obstajata javna ključa obeh udeležencev transakcije, program `HE_Client` izvede enkripcijo zneska (enkrat z javnim ključem začetnika transakcije, drugič z javnim ključem prejemnika zneska)

in oba tajnopisa posreduje razširitvi. Skripta za spreminjanje vsebine spletne strani tajnopisa zapiše v skrita gradnika spletne strani. Sprememba vsebine teh gradnikov sproži pošiljanje tajnopisa in ostalih podatkov o transakciji na strežnik spletne banke, ki poskrbi za izvedbo homomorfne transakcije.

Zgradbo odjemalca naše programske rešitve z razširitvijo za brskalnik Chrome, vključno s sestavnimi deli razširitve in tokom podatkov za opisan primer prikazuje slika 5.6. Pri dekripciji, ki se izvede v okviru prikaza trenutnega stanja ali pregleda zgodovine stanj na transakcijskem računu, je tok podatkov enak, le da se začne z branjem tajnopisa in zaključi s prikazom čistopisa na spletni strani.



Slika 5.6: Arhitekturna zasnova odjemalca pri pristopu z razširitvijo za brskalnik Chrome. Slika prikazuje tok podatkov od spletne strani do lastnega programa in nazaj za primer enkripcije, ki se izvede za podporo homomorfnim transakcijam.

5.5 Pristop z dodatnim posvečenim strežnikom

Pristopa, opisana v razdelkih 5.3 in 5.4, imata dve veliki pomanjkljivosti:

1.) **Omejitev na operacijski sistem iz družine Linux**. Pristopa od odjemalca zahtevata uporabo operacijskega sistema iz družine Linux, saj je

knjižnica HElib trenutno kompatibilna le s to družino operacijskih sistemov. To močno omeji nabor odjemalcev, ki omenjena pristopa lahko uporablja, saj ima po statistiki iz leta 2016 le okrog šest odstotkov odjemalcev nameščene operacijske sisteme Linux [43].

2.) **Obremenitev naprave odjemalca**. Noben od pristopov ni primeren za uporabo na računsko manj zmogljivih napravah (npr. mobilnih telefonih ali tabličnih računalnikih), saj enkripcija in dekripcija zahtevata znatno količino procesiranja na strani odjemalca.

Kot alternativo pristopoma iz razdelkov 5.3 in 5.4, ki opisanih pomanjkljivosti nima, predlagamo *pristop z dodatnim posvečenim strežnikom.* Če v arhitekturo uvedemo dodatno komponento - posvečeni strežnik -, lahko odjemalec na napravi s poljubnim operacijskim sistemom poganja zgolj spletni brskalnik in obenem ostane neobremenjen z režijskim delom, ki je posledica enkripcije in dekripcije. Posvečeni strežnik poganja operacijski sistem Linux, zato lahko zažene programe, ki uporabljajo knjižnico HElib, obenem pa je bolj zmogljiv od povprečne naprave odjemalca oblačnih storitev.

Posvečeni strežnik odjemalca razbremeni tako, da na njegovo zahtevo izvede operacije, ki se morajo v pristopih 5.3 in 5.4 izvesti na napravi odjemalca. To vključuje logiko programa v C++ za enkripcijo in dekripcijo podatkov, zato smo ta program v okviru pristopa z dodatnim posvečenim strežnikom poimenovali `HE_RemoteClient` (t. i. *odjemalec na oddaljeni napravi*).

5.5.1 Lokacija posvečenega strežnika

Opisan pristop odjemalčevo napravo razbremeni tako, da operaciji enkripcije in dekripcije preloži na posvečeni strežnik. Slednje pomeni, da mora imeti posvečeni strežnik dostop tako do javnih kot tudi do zasebnih ključev vseh odjemalcev, ki jih streže. Ker posvečeni strežnik upravlja z zasebnimi ključi, je za kakršno koli realno postavitev smiselno, da deluje v lokalnem (zaupanja vrednem) omrežju uporabnika ali organizacije in ne v računalniškem oblaku.

5.5.2 Upravljanje z zasebnimi ključi

Dodatni posvečeni strežnik v opisani rešitvi nastopa kot zaupanja vredna tretja entiteta (angl. *trusted third party*), saj mu odjemalci zaupajo upravljanje z njihovimi zasebnimi ključi. Vsak odjemalec od posvečenega strežnika v zameno pričakuje:

- da bo do njegovega zasebnega ključa posvečeni strežnik dostopal samo na njegovo zahtevo (tj. za izvedbo enkripcije ali dekripcije) in
- da njegovega zasebnega ključa posvečeni strežnik ne bo razkril nikomur, razen po potrebi njemu samemu.

Spomnimo se, da smo v razdelku 2.2.3 poudarili, da preferiramo rešitve, ki od odjemalca zahtevajo čim manjše zaupanje, vendar smo govorili o zaupanju v ponudnika oblačnih storitev. Zavedati se moramo, da v naši postavitvi posvečeni strežnik ni del sistema računalniškega oblaka, ampak je last posameznika ali organizacije, ki svoje podatke pošlje v obdelavo v računalniški oblak. Zaradi slednjega se nam zdi predlagan pristop s posvečenim strežnikom kljub temu sprejemljiv, če uporablja ustrezne mehanizme za avtentikacijo, avtorizacijo in revidiranje. Implementacija omenjenih mehanizmov presega obseg našega dela, zato zgolj predpostavimo njihov obstoj v realni postavitvi, sicer pa se z njimi ne ukvarjamo.

5.5.3 Spletna aplikacija na strani posvečenega strežnika

Posvečeni strežnik mora odgovarjati na zahteve spletnega brskalnika, ki ga uporablja končni uporabnik spletne banke, zato smo za njegovo realizacijo izdelali še eno spletno aplikacijo.

Za implementacijo spletne aplikacije posvečenega strežnika smo uporabili iste tehnologije kot pri aplikaciji spletne banke. Gre za zelo preprosto spletno aplikacijo, ki ima uporabniški vmesnik definiran samo za potrebe enostavnejšega upravljanja z datotekami, in sicer za prenos javnega in zasebnega ključa s posvečenega strežnika (javni ključ mora namreč odjemalec poslati spletni banki) in za nalaganje datoteke s tajnopisom zgodovine stanj transakcijskega računa na posvečeni strežnik pred dekripcijo. Razen tega pa spletna aplikacija posvečenega strežnika navzven izpostavlja zgolj programski vmesnik s tremi končnimi točkami, in sicer za inicializacijo (generiranje ključev), enkripcijo in dekripcijo, ter na odjemalčeve zahteve odgovarja s sporočili v formatu JSON.

5.5.4 Komunikacija med odjemalcem in posvečenim strežnikom

Glede na funkcionalnost posvečenega strežnika, do katere želi dostopati odjemalec, z le-tem komunicira na dva različna načina - neposredno ali posredno.

Neposredna komunikacija

Pri neposredni komunikaciji končni uporabnik spletne banke obišče spletni naslov aplikacije posvečenega strežnika in bodisi prenese svoj javni ali zasebni ključ na svojo napravo bodisi na posvečeni strežnik naloži javni ključ drugega uporabnika. Ta način komunikacije s posvečenim strežnikom je torej namenjen upravljanju s kriptografskimi ključi.

Posredna komunikacija

Pri posredni komunikaciji končni uporabnik ne obišče spletne strani aplikacije posvečenega strežnika v spletnemu brskalniku, ampak komunikacijo s posvečenim strežnikom začne skripta spletne banke. Ta način komunikacije uporabljamo za izvedbo enkripcije in dekripcije, ki zahtevata klic funkcij programa `HE_RemoteClient`. Da lahko končni uporabnik pridobi podatek, ki zahteva izvedbo omenjenih operacij, se mora izvesti naslednje zaporedje korakov:

1.) Uporabnik obišče ustrezno spletno stran aplikacije spletne banke, s tem pa se v brskalnik naloži tudi pripadajoča skripta s kodo v programskem jeziku JavaScript.

2.) Ob uporabnikovi interakciji z gradniki spletne strani (npr. ob kliku na gumb) skripta v jeziku JavaScript prebere vhodne podatke s spletne strani - bodisi tajnopis podisi čistopise - in pošlje asinhroni zahtevek AJAX (angl. *Asynchronous JavaScript and XML*) na ustrezno končno točko posvečenega strežnika.

3.) Spletna aplikacija posvečenega strežnika po prejemu zahtevka pokliče ustrezno funkcijo programa `HE_RemoteClient`, pri čemer funkcijo pokliče iz kode v programskem jeziku Python na enak način, kot spletna aplikacija banke kliče funkcije programa `HE_Server`.

4.) Funkcija programa `HE_RemoteClient` se zaključi in vrne rezultat spletni aplikaciji posvečenega strežnika. Slednja sporoči rezultat klicatelju (skripti spletne banke) v obliki sporočila v formatu JSON.

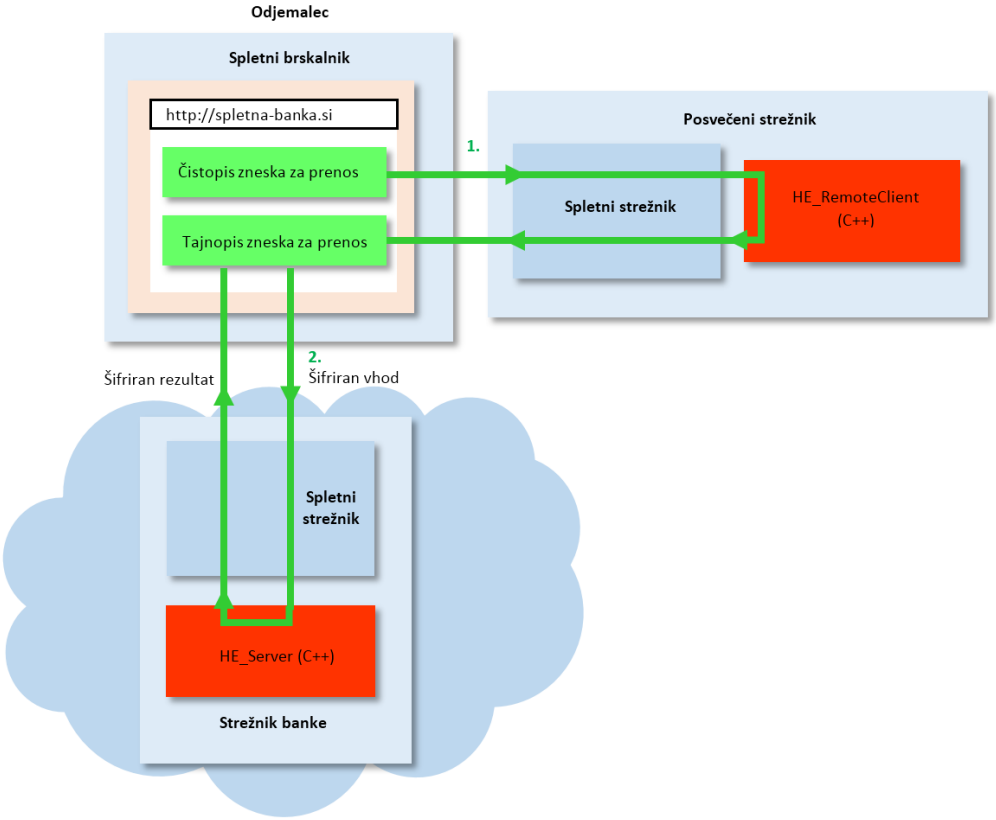
6.) Skripta spletne banke ob prejemu odgovora posvečenega strežnika uporabi rezultat dane funkcije za spremembo vsebine spletne strani (npr. prikaz dekriptiranega zneska), ki jo je v spletnem brskalniku obiskal končni uporabnik.

Tok podatkov pri posredni komunikaciji med odjemalcem in posvečenim strežnikom za primer izvedbe homomorfne transakcije prikazuje slika 5.7.

Pošiljanje asinhronih zahtevkov v drugo domeno

Zaporedje korakov pri posredni komunikaciji med odjemalcem in posvečenim strežnikom predvideva pošiljanje asinhronih zahtevkov iz kode, ki izvira iz domene spletne banke, v domeno posvečenega strežnika. Slednje je v današnjih spletnih brskalnikih privzeto onemogočeno, saj se brskalniki pri izvajanju skript ravnajo v skladu s t. i. *politiko istega izvora* (angl. *same-origin policy*). V skladu s to politiko spletni brskalnik dovoli zgolj nalaganje spletnih virov, ki prihajajo iz iste spletne domene kot vir, ki ga je zahteval končni uporabnik. Gre za varnostni mehanizem, s katerim brskalniki morebitnim napadalcem preprečujejo izvedbo podtaknjene zlonamerne programske kode.

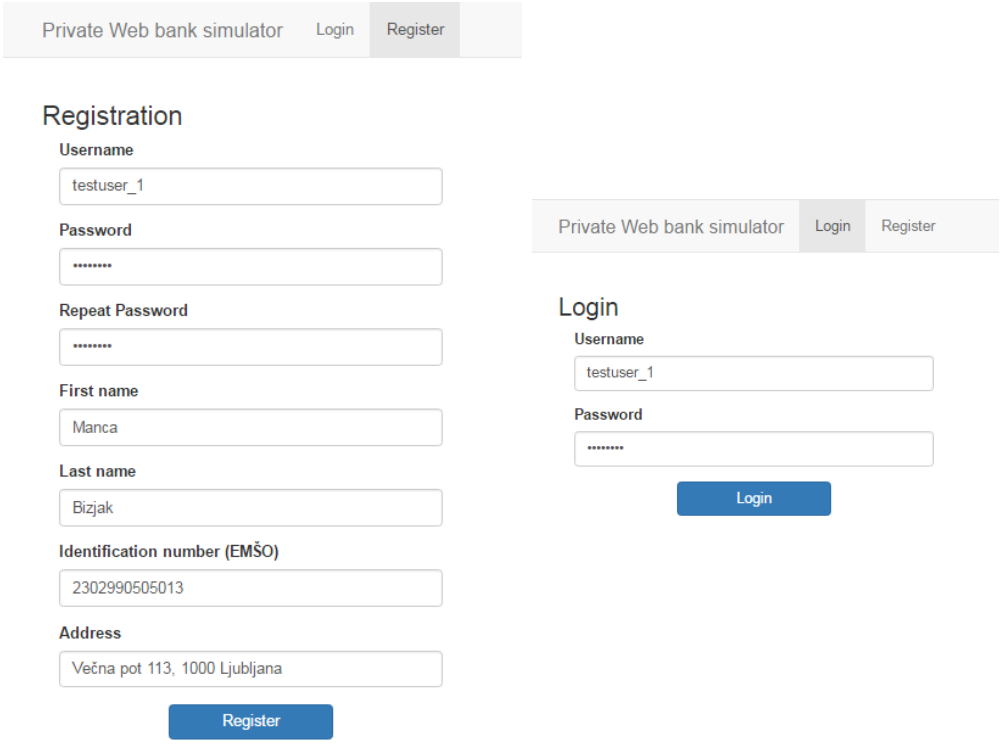
Da omogočimo opisano komunikacijo med odjemalcem in posvečenim strežnikom, moramo na strani posvečenega strežnika dovoliti obdelavo zahtevkov, ki prihajajo iz domene spletne banke, s pomočjo t. i. *meddomenskega deljenja virov* (*cross-origin resource sharing*, CORS). Prav tako mora tudi spletna banka poznati naslov posvečenega strežnika in končne točke njegovega programskega vmesnika. To pomeni, da je spletna banka tesno sklopljena s posvečenim strežnikom.



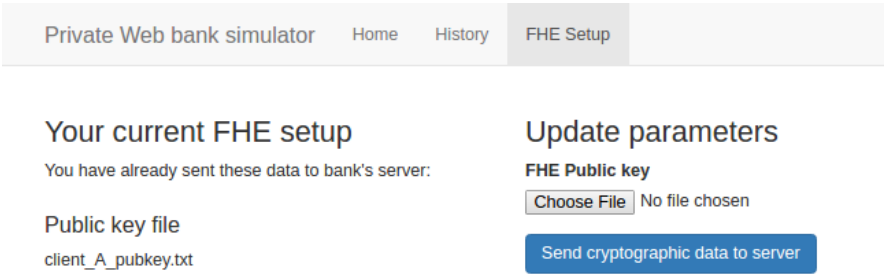
Slika 5.7: Arhitekturna zasnova rešitve za uporabo polne homomorfne enkripcije pri pristopu z dodatnim posvečenim strežnikom. Prikazani primer prikazuje tok podatkov pri posredni komunikaciji med odjemalcem in posvečenim strežnikom, in sicer na primeru zahteve za izvedbo homomorfne transakcije. Posvečeni strežnik na odjemalčevo zahtevo zakriptira znesek, za tem pa ga odjemalec skupaj z ostalimi podatki za izvedbo transakcije pošlje na strežnik banke.

5.6 Uporabniški vmesnik

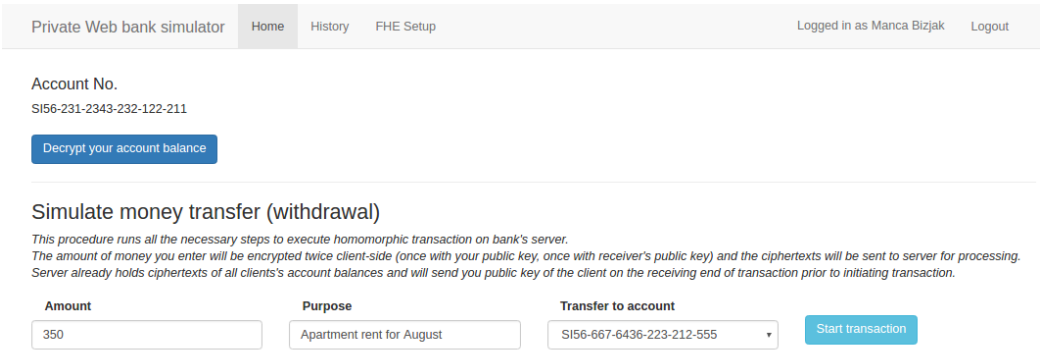
V nadaljevanju podajamo še zaslonske posnetke spletnih strani naše aplikacije spletne banke. Uporabniški vmesnik je pri obeh implementiranih pristopih na strani odjemalca enak. Slika 5.8 prikazuje spletna obrazca za registracijo in prijavo uporabnika v spletno banko. Na sliki 5.9 vidimo izsek spletne strani na kateri uporabnik iz datotečnega sistema na svoji napravi izbere svoj javni ključ in ga pošlje spletni banki. Domačo stran, na kateri uporabnik za pregled trenutnega stanja na transakcijskem računu sproži dekripcijo, ali pa banki pošlje zahtevek za izvedbo bančne transakcije, prikazuje slika 5.10. Primer prikaza zgodovine bančnih transakcij na transakcijskem računu uporabnika takoj po dekripciji prikazuje slika 5.11.



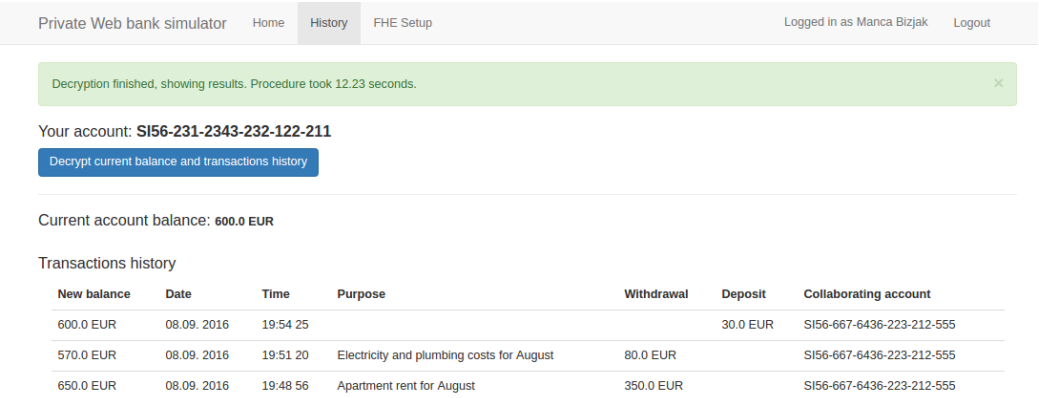
Slika 5.8: Obrazca za registracijo (levo) in prijavo uporabnika (desno) v aplikaciji spletne banke.



Slika 5.9: Spletna stran za nastavitev javnega ključa prijavljenega uporabnika v aplikaciji spletne banke. Preden lahko uporabnik pregleda stanje svojega transakcijskega računa ali izda zahtevo za izvedbo bančne transakcije, mora banki poslati svoj javni ključ.



Slika 5.10: Domača stran prijavljenega uporabnika v spletni aplikaciji banke. Uporabnik mora za prikaz trenutnega stanja na transakcijskem računu s klikom na gumb sprožiti dekripcijo. Prav tako lahko izda zahtevo za izvedbo bančne transakcije.



Slika 5.11: Pregled stanja na transakcijskem računu in zgodovine bančnih transakcij za prijavljenega uporabnika spletne banke. Preden lahko uporabnik v spletni aplikaciji vidi trenutno stanje na računu in tabelo z zgodovino transakcij, mora s klikom na gumb sprožiti dekripcijo.

Poglavje 6

Ovrednotenje praktične uporabnosti

Preden lahko sodimo o praktični uporabnosti polne homomorfne enkripcije na podlagi naših implementacij, jih moramo preizkusiti v realni postavitvi. To poglavje je namenjeno razpravi o praktični uporabnosti implementiranih rešitev za uporabo PHE preko spletne aplikacije, ki izhajajo tako iz naših izkušenj s programsko knjižnico HElib kot tudi iz konkretnih meritev. Implementirane rešitve ovrednotimo s pomočjo mer, ki smo jih definirali v poglavju 4.

6.1 Testno okolje

Za testiranje smo imeli na razpolago zasebni računalniški oblak, na katerem je bila nameščena platforma OpenStack, ki omogoča upravljanje z oblačno infrastrukturo. V njem smo postavili virtualno napravo z operacijskim sistemom Linux Ubuntu 14.04, osmimi procesorskimi jedri in 8 GB delovnega pomnilnika. Virtualna naprava je poganjala spletni strežnik, ki je odjemalcem stregel spletno aplikacijo banke.

Za testiranje pristopa z razširitvijo za brskalnik Chrome smo uporabili računalnik z operacijskim sistemom Linux Ubuntu 14.04, štirijedrnim procesorjem in 8 GB delovnega spomina, na katerem sta tekla spletni brskalnik in razširitev.

Enako zmogljiv računalnik smo uporabili tudi pri testiranju pristopa z doda-

tnim posvečenim strežnikom, le da smo v tem primeru spletni brskalnik zaganjali iz operacijskega sistema Windows 7. Poleg tega smo morali postaviti še dodatni posvečeni strežnik, ki smo ga namestili na isti računalnik, vendar v virtualno napravo z gostujočim operacijskim sistemom Linux Ubuntu 14.04. Naša postavitev torej predvideva, da sta odjemalec in posvečeni strežnik, kot ju prikazuje slika 5.7, fizično del istega računalniškega sistema.

6.1.1 Vrednosti kriptografskih parametrov

Preostanek tega poglavja v naših nizkonivojskih implementacijah predpostavlja vrednosti kriptografskih parametrov knjižnice HElib, navedene v tabeli 6.1, razen, če ni navedeno drugače.

Parameter	Vrednost
k	80
p	2
r	32
L	12

Tabela 6.1: Izbrane vrednosti kriptografskih parametrov knjižnice HElib.

Za parameter varnosti k smo uporabili privzeto (priporočeno) vrednost. Vrednost parametra L smo določili s poskušanjem, p pa ima lahko v trenutni različici knjižnice HElib zgolj vrednost 2. Parameter r ima vrednost 32, saj smo na ta način podprli 32-bitno celoštevilsko aritmetiko.

6.2 Produkcijska zrelost

Naša spletna banka, posledično pa tudi oba implementirana pristopa na strani odjemalca, služijo zgolj za demonstracijo uporabe PHE preko spleta, zato je vrednotenje nivoja njihove produkcijske zrelosti brezpredmetno. Nasprotno pa lahko komentiramo produkcijsko zrelost trenutno najbolj zrele knjižnice za podporo PHE, HElib.

Čeprav HElib predstavlja pomemben korak k praktični uporabi polne homomorfne enkripcije, sama po sebi razvijalcem oblačnih aplikacij ne ponuja vmesnika

za njeno enostavno uporabo. Če želi programer uporabiti funkcionalnosti HElib v spletni aplikaciji, se mora poleg z aplikacijsko logiko v praviloma visokonivojskem programskem jeziku, v katerem je napisana spletna aplikacija, ukvarjati še z implementacijo homomorfnega procesiranja v nizkonivojskem C++.

Trenutna verzija knjižnice pa ima poleg slednjega še številne praktične omejitve, s katerimi se običajne aplikacije (tj. aplikacije, ki obdelujejo čistopise), praviloma ne srečujejo, in sicer:

- **ni podpore za enkripcijo decimalnih števil in nizov**. V splošnem pomanjkanje podpore za homomorfno procesiranje nizov ni tako kritično, saj je procesiranje nizov v praksi manj pogosto kot procesiranje številskih podatkov. Po drugi strani pa je podpora homomorfnemu procesiranju decimalnih števil nepogrešljiva, zlasti v domeni finančnih aplikacij.

- **Ni podpore za operacijo deljenja**. Operacijo deljenja sicer lahko prevedemo na množenje z obratno vrednostjo, vendar v tem primeru v splošnem pridelamo decimalno število. Slednje pa nas zopet privede do dejstva, da s tajnopisi ne moremo predstaviti decimalnih števil.

Zgornje omejitve izhajajo iz pomanjkanja vgrajene podpore za podatkovne tipe in operacije, ki jih lahko izvajamo nad tajnopisi. Očitno je torej, da uporaba knjižnice HElib programerju dela ne poenostavi, saj mu za homomorfno procesiranje ne nudi uporabe sicer samoumevnih podatkovnih tipov in operacij.

Tudi če bi bili na račun povečanja zasebnosti pripravljeni uporabljati v splošnem že tako neučinkovite rešitve, opisane pomanjkljivosti še dodatno omejujejo nabor aplikacij, pri katerih bi bila implementacija podpore za PHE s pomočjo knjižnice HElib smiselna.

6.2.1 Večnitno izvajanje

Podpora večnitnemu izvajanju v HElib je z vidika produkcijske zrelosti ključna, saj lahko večnitno izvajanje pohitri programe tako na strani odjemalca kot tudi strežnika. Avtorji knjižnice HElib navajajo do 1.5-kratno pohitritev nizkonivojskih transformacij, realiziranih s pomočjo programske knjižnice NTL. Slednjih naj bi bilo v tipičnem programu, ki uporablja HElib, okrog 40 odstotkov [24], razen

tega pa v dokumentaciji ni natančnejših podatkov. Kljub temu v nobenem od naših programov nismo opazili navedenih pohitritev v primerjavi z implementacijami brez večnitnega izvajanja - povprečni izvajalni časi vseh funkcij v programih C++ tako na strani odjemalca (`HE_Client` in `HE_RemoteClient`) kot tudi strežnika (`HE_Server`) so bili kvečjemu le za nekaj odstotkov krajši, tako da so bile pohitritve zanemarljive. Ker večnitne različice naših programov niso prinesle opaznih pohitritev, se v preostanku tega poglavja z njimi ne ukvarjamo več. Nadaljnje meritve zato temeljijo na enonitnih različicah programov v C++.

Za resno praktično uporabo je bolj kot pohitritev rutin na nivoju knjižnice NTL pomembna pohitritev postopka samozagona, ki ga mora izvesti strežnik. Samozagon je namreč najpomembnejše ozko grlo polnih homomorfnih enkripcijskih shem. Ta pohitritev naj bi bila z večnitnim izvajanjem v HElib več kot 10-kratna. Naša implementacija programa `HE_Server` samozagona sicer ne uporablja zaradi skrbno izbrane vrednosti parametra L in obvladljivega nivoja šuma v tajnopisih - slednji je posledica uporabe zgolj homomorfnih seštevanj, odštevanj in premikov, ne pa tudi množenj.

Zaradi pomanjkanja programerjem prijaznega programskega vmesnika in zgolj eksperimentalne podpore večnitnemu izvajanju zaključujemo, da je nivo produkcijske zrelosti knjižnice HElib - sicer trenutno najboljše alternative za realizacijo PHE - nizek.

6.3 Količina omrežnega prometa

V naših implementacijah preko omrežja pogosto prenašamo tajnopise, pa tudi (predvsem javne) kriptografske ključe. Količina omrežnega prometa, ki je posledica komunikacije med odjemalcem in strežnikom (v primeru pristopa z dodatnim posvečenim strežnikom pa tudi med odjemalcem in posvečenim strežnikom) ter ni namenjena prenosu tajnopisov ali ključev, je zanemarljiva. Količina dodatnega omrežnega prometa, ki je posledica uporabe PHE, je torej v veliki meri odvisna od velikosti in števila tajnopisov ter kriptografskih ključev.

Opazili smo, da so tajnopisi in kriptografski ključi, s katerimi rokujejo naše programske rešitve, razmeroma veliki glede na njihove velikosti v uveljavljenih enkripcijskih shemah. Naše analize so pokazale, da na velikost tajnopisov in krip-

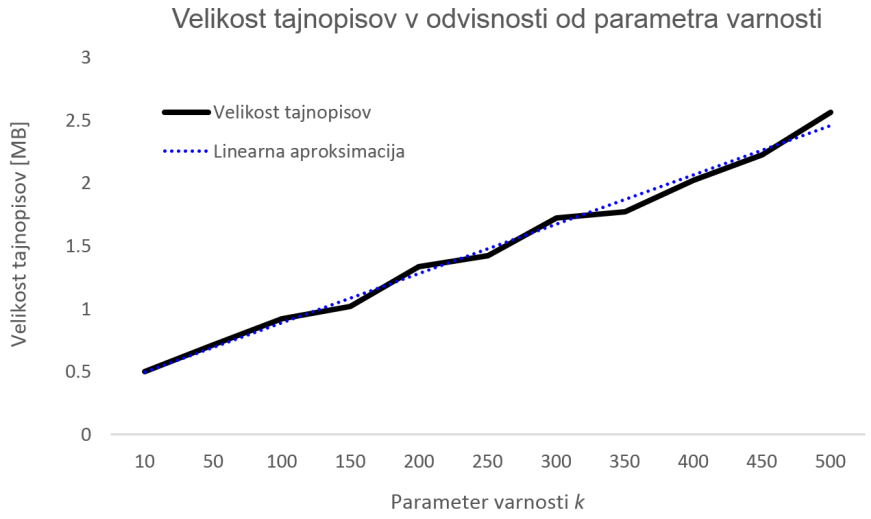
tografskih ključev sicer vpliva več različnih kriptografskih parametrov knjižnice HElib, vendar pa ima nanje največji vpliv parameter varnosti k. V nadaljevanju zato navajamo le rezultate analiz pri variaciji parametra varnosti.

6.3.1 Tajnopisi

Naša implementacija med odjemalcem in strežnikom predvideva prenašanje tajnopisov z zgodovino stanj na transakcijskem računu odjemalcev in tajnopisov z zneski za prenos v okviru bančne transakcije.

Velikost tajnopisov

Analizirali smo povprečno velikost tajnopisov, ki se v naših programskih rešitvah prenašajo preko omrežja. Ugotovili smo, da obstaja izrazito linearna zveza med velikostjo tajnopisov in parametrom varnosti k. Rezultate naših meritev prikazuje slika 6.1, ki prikazuje graf velikosti tajnopisov v odvisnosti od parametra varnosti k.



Slika 6.1: Spreminjanje velikosti tajnopisov v odvisnosti od varnostnega parametra k. Prekinjena modra črta prikazuje premico, ki najbolje aproksimira podatke.

Iz grafa lahko razberemo, da je velikostni red tajnopisov pri analiziranih vrednostih parametra k enak 1 MB. Pri $k = 80$, ki je bila tipična vrednost parametra varnosti v nadaljnjih eksperimentih, so bili tajnopisi veliki okrog 800 KB.

Število prenosov tajnopisov

Ob obisku domače strani spletna banka uporabniku pošlje tajnopis z zgodovino stanj na njegovem transakcijskem računu. Če pa želi uporabnik banke izvesti transakcijo, mora do strežnika spletne banke sam prenesti dva tajnopisa zneska za prenos v okviru transakcije (en je kriptiran z lastnim javnim ključem, drugi z javnim ključem prejemnika). V prvem primeru se torej preko omrežja prenese en tajnopis, v drugem pa dva.

Zgornji podatki za število prenešenih tajnopisov pravzaprav veljajo samo za pristop, kjer na strani odjemalca uporabljamo razširitev za brskalnik Chrome. Ob uporabi pristopa z dodatnim posvečenim strežnikom je ne glede na želeno akcijo (prikaz zgodovine bančnih transakcij ali zahteva za izvedbo transakcije) skupno število tajnopisov, ki se prenese preko omrežja, podvojeno. Za dekripcijo zgodovine stanj mora namreč odjemalec tajnopis, ki ga je dobil od spletne banke, posredovati posvečenemu strežniku. Prav tako pa mora posvečeni strežnik pred izvedbo transakcije odjemalcu posredovati dva tajnopisa, da ju le-ta lahko posreduje spletni banki.

6.3.2 Kriptografski ključi

V naših implementacijah se preko omrežja praviloma prenašajo le javni ključi. Pri uporabi pristopa z dodatnim posvečenim strežnikom ima uporabnik sicer možnost prenosa svojega zasebnega ključa s posvečenega strežnika na lastno napravo, vendar zaradi njene redkosti te operacije ne obravnavamo in se v nadaljevanju posvetimo javnim ključem.

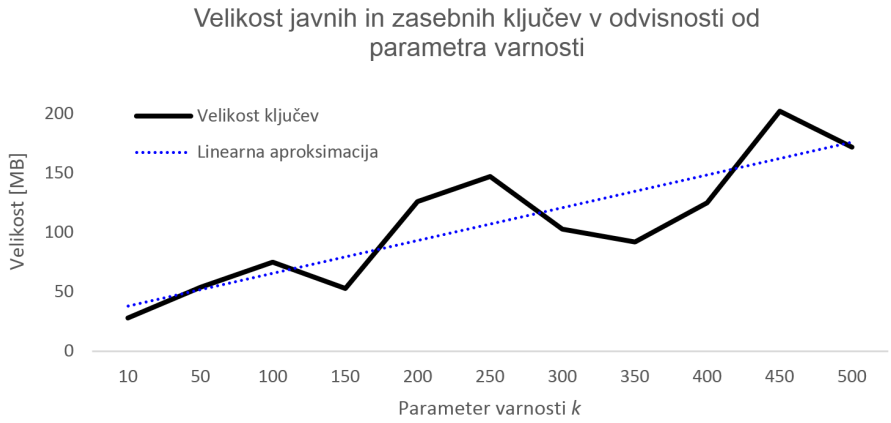
Velikost kriptografskih ključev

Velikost kriptografskih ključev danes v praksi praviloma navajamo v številu bitov, slednja pa ne glede na uporabljeno enkripcijsko shemo v tipičnih primerih uporabe redko preseže velikost 4096 bitov (512 B). Pri izbranih vrednostih kriptografskih

parametrov je velikostni red ključev, s katerimi rokujejo naše programske rešitve, 100 MB. To pomeni, da so ključi v našem primeru reda 10^5-krat večji od običajnih.

Takšne velikosti kriptografskih ključev so posledica uporabe primitivov asimetrične *kriptografije, ki temelji na idealnih mrežah* (angl. *ideal lattice-based cryptography*), ki so v ozadju vseh trenutno znanih polnih homomorfnih enkripcijskih shem. Kriptografija idealnih mrež je zelo aktualna, saj je dokazano odporna na napade z algoritmi kvantnih računalnikov, vendar pa so vsem njenim različicam skupni (včasih celo neobvladljivo) veliki kriptografski ključi [8, 28].

Tudi za kriptografske ključe smo preverili zvezo med parametrom varnosti k in njihovo velikostjo. Graf odvisnosti velikosti kriptografskih ključev od parametra k prikazuje slika 6.2. Iz grafa je razvidno, da se za dovolj velike spremembe parametra k velikosti ključev povečujejo, vendar pa zveza ni tako izrazito linearna kot pri tajnopisih (slika 6.1). Na primer povečanje parametra k zgolj za vrednost 50 se lahko odraža tako v zmanjšanju kot tudi v povečanju velikosti kriptografskih ključev.



Slika 6.2: Spreminjanje velikosti kriptografskih ključev v odvisnosti od varnostnega parametra k. Zasebni ključi so bili konsistentno za 2 MB večji od javnih, tako da je velikostni red obojih enak. Na grafu je z modro prekinjeno črto predstavljena tudi premica, ki najbolje aproksimira naše meritve.

Omenjena zveza med velikostjo kriptografskih ključev in parametrom k je intuitivna, saj velikost kriptografskih ključev pogojuje nivo varnosti - daljši kot so

ključi, težje jih je razbiti, posledično pa so bolj varni.

Število prenosov kriptografskih ključev

Uporabnik mora ob prvi uporabi spletne aplikacije prenesti svoj javni ključ na strežnik banke. Obenem pa mora iz strežnika banke pred vsakim zahtevkom za izvedbo transakcije prenesti javni ključ prejemnika transakcije, če le-tega še nima shranjenega lokalno. Če torej zanemarimo enkraten prenos javnega ključa uporabnika na strežnik spletne banke, je število prenešenih kriptografskih ključev v najslabšem primeru enako številu bančnih transakcij. To se zgodi, če mora odjemalec pred vsako transakcijo iz strežnika banke prenesti javni ključ prejemnika.

Zgornje zopet velja za primer, ko odjemalec uporablja pristop z razširitvijo za brskalnik Chrome. Število prenosov kriptografskih ključev v primeru uporabe pristopa z dodatnim posvečenim strežnikom je, podobno kot pri številu prenosov tajnopisov, podvojeno. Pred prvim prenosom uporabnikovega ključa na strežnik banke mora namreč uporabnik slednjega najprej prenesti na svojo napravo s posvečenega strežnika. Podobno pa mora vsak tuji ključ uporabnika na prejemni strani transakcije po prenosu iz strežnika banke prenesti še na posvečeni strežnik.

Količina omrežnega prometa, ki nastane kot posledica uporabe PHE, je relativno velika, in sicer predvsem na račun prenašanja javnih ključev odjemalcev na prejemni strani transakcije. Slednji so namreč približno stokrat večji od tajnopisov. S stališča proizvedenega omrežnega prometa je uporaba pristopa z razširitvijo za brskalnik Chrome bolj ugodna, saj zahteva prenašanje dvakrat manjše količine podatkov kot pristop z dodatnim posvečenim strežnikom.

6.4 Poraba računalniških virov v oblaku

Na porabo računalniških virov v oblaku neposredno vplivata spletna aplikacija banke in strežniški program za homomorfno procesiranje `HE_Server`. Pri tem predpostavljamo, da je prispevek k porabi virov zaradi spletne aplikacije konstanten, in se osredotočimo zgolj na dodatno obremenitev računalniških virov zaradi uporabe PHE pri izvajanju bančnih transakcij.

Porabo računalniških virov smo spremljali na nivoju strežnika spletne banke v

računalniškem oblaku. Pri tem smo si pomagali z orodjem za analitiko računalniških virov New Relic [31], ki nam je omogočilo zajem podatkov o trenutni in povprečni obremenitvi procesorja (angl. *load average*), porabi delovnega pomnilnika ter trdega diska.

Vse meritve porabe računalniških virov, ki bodo predstavljene na grafih v nadaljevanju, smo zajeli sočasno. V postopku testiranja smo v aplikaciji spletne banke najprej večkrat zaporedoma sprožili zahtevo za izvedbo transakcije z uporabniškega računa ene stranke. Za tem smo nekaj minut počakali in se prijavili v spletno banko kot z uporabniškim računom druge stranke ter še z njenega računa sprožili nekaj zaporednih zahtev za izvedbo transakcije. Postopek smo nato večkrat ponovili.

Strežnik banke je v danem trenutku obdeloval kvečjemu eno zahtevo za izvedbo transakcije.

6.4.1 Obremenitev procesorja

Trenutna obremenitev

Virtualna naprava, na kateri smo poganjali spletno banko, je imela 8 procesorskih jeder. 100-odstotno obremenitev procesorja bi zato v našem primeru dosegli, če bi vsa jedra računala s polno kapaciteto. Eno procesorsko jedro torej pri polni obremenjenosti prispeva največ 12.5 odstotkov k skupni obremenitvi procesorja. Spomnimo se, da vsa koda oblačne aplikacije teče v eni programski niti, kar pomeni, da se v danem trenutku izvaja kvečjemu na enem procesorskem jedru, in zato ob odsotnosti izvrševanja drugih procesorsko zahtevnih programov ne pričakujemo obremenitve, ki bi presegla 12.5 odstotkov.

Ob izvrševanju funkcije programa `HE_Server`, v okviru katere izvedemo homomorfno transakcijo, se je obremenitev procesorja v povprečju dvignila za 6-8 odstotkov. Če bi spletni strežnik sočasno obdeloval zahtevi za izvedbo transakcij dveh različnih odjemalcev, bi dosegli polno kapaciteto procesorskega jedra. Rezultate meritev porabe procesorja podajamo na zgornjem delu slike 6.3, kjer vrhovi sovpadajo s procesorsko zahtevnimi deli programske kode za izvajanje homomorfnih bančnih transakcij.

Povprečna obremenitev

Poleg trenutne porabe oz. obremenitve procesorja pa nas v praksi pogosto zanima tudi njegova povprečna obremenitev. Povprečna obremenitev pomeni število aktivnih in čakajočih procesov v danem časovnem intervalu. Za posamezno procesorsko jedro povprečna obremenitev lahko zavzame vrednosti med 0 (neobremenjeno jedro) in 1 (jedro računa s polno kapaciteto), pri čemer v praksi preferiramo čim nižje vrednosti. Visoke vrednosti - na primer nad 0.7 za posamezno jedro - so indikator preobremenjenosti.

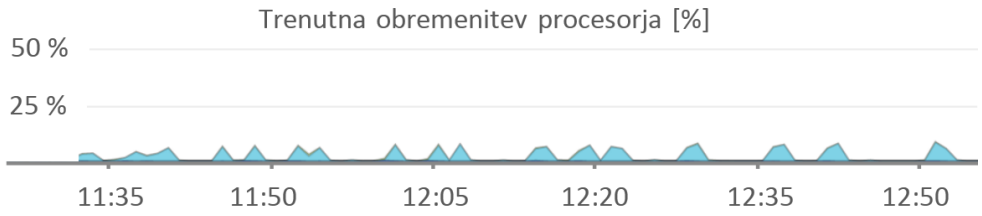
Graf na spodnjem delu slike 6.3 prikazuje povprečno obremenitev procesorja v postopku testiranja, pri čemer so povprečja izračunana na intervalu ene minute (t. i. *load-1*). Iz grafa je razviden izrazit porast obremenjenosti, ki se v času izvajanja programa `HE_Server` med izvedbo bančnih transakcij iz zelo nizke vrednosti (okrog 0.01) povzpne na 0.5 ali celo 0.6.

6.4.2 Poraba delovnega pomnilnika

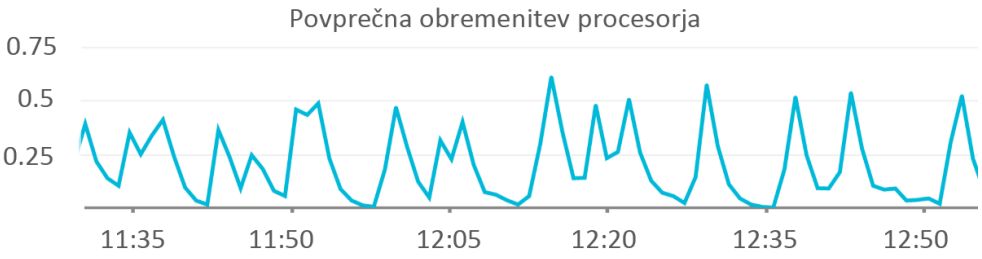
Porabo delovnega pomnilnika v času obdelave zaporednih zahtev za izvedbo bančnih transakcij prikazuje graf na sliki 6.4. Na račun izvedbe bančnih transakcij se je poraba delovnega pomnilnika v povprečju povzpela na okrog 5 odstotkov, kar pri kapaciteti pomnilnika 8 GB pomeni približno 400 MB. V programu `HE_Server` približno polovico od tega prispeva vzdrževanje javnih ključev uporabnikov, katerih računa sodelujeta v transakciji. Kot smo videli v razdelku 6.3.2, sta namreč slednja velikostnega reda 100 MB. Glede na to, da je tako povišanje posledica obdelave ene same zahteve za izvedbo transakcije naenkrat, je poraba delovnega pomnilnika relativno visoka.

6.4.3 Poraba prostora na disku

Zaradi uporabe PHE mora virtualna naprava, ki poganja strežnik spletne banke, za vsakega uporabnika banke vzdrževati njegov javni ključ in tajnopis z zgodovino stanj na njegovem računu. Če je transakcijski račun posameznega uporabnika udeležen v velikem številu transakcij, se lahko zgodi, da moramo zanj vzdrževati več tajnopisov. Kljub temu pa je za porabo prostora na disku pomembnejši doprinos zaradi javnih ključev, saj so z velikostnim redom 100 MB pri uporabljenih

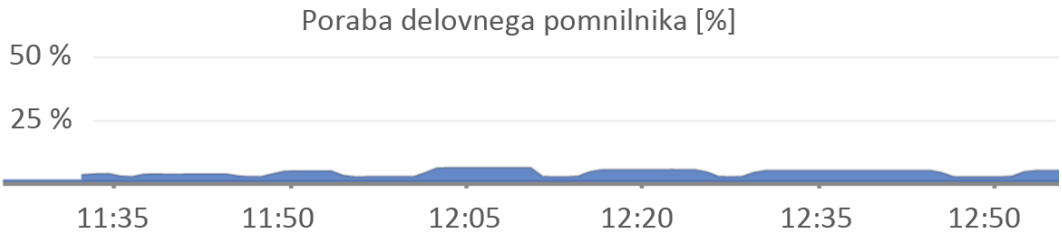


(a)



(b)

Slika 6.3: Trenutna (a) in povprečna (b) obremenitev procesorja, ki je posledica uporabe PHE na strani strežnika spletne banke. Iz grafov lahko razberemo, kdaj je strežnik izvrševal funkcijo za izvedbo bančnih transakcij, saj je bila v nasprotnem primeru obremenitev zanemarljiva. V danem trenutku je strežnik obdeloval kvečjemu eno zahtevo za izvedbo transakcije.



Slika 6.4: Poraba delovnega pomnilnika na strežniku spletne banke, ki je posledica uporabe PHE. V danem trenutku je strežnik obdeloval kvečjemu eno zahtevo za izvedbo transakcije.

kriptografskih parametrih približno stokrat večji od tajnopisov.

Dodatna poraba prostora na disku, ki nastane zaradi uporabe PHE v spletni aplikaciji, je torej premo sorazmerna s številom uporabnikov spletne banke. Pri

n uporabnikih, velikosti javnih ključev k MB in velikosti tajnopisov t MB znaša približno $n * (k + t)$ MB. Pri našem izboru kriptografskih parametrov to ustreza $100 * n + n$ MB.

6.5 Transparentnost za uporabnika

6.5.1 Vpletenost uporabnika

Z izjemo količine omrežnega prometa so bile do sedaj vse mere, s katerimi smo vrednotili praktično uporabnost naših implementacij, neodvisne od uporabe konkretnega pristopa na strani odjemalca. Nasprotno pa je z vidika vpletenosti uporabnika v operacije za podporo PHE izbira pristopa na strani odjemalca (tj. pristopa z razširitvijo za brskalnik Chrome ali pristopa z dodatnim posvečenim strežnikom) ključna za nivo transparentnosti za uporabnika. V nadaljevanju zato vpletenost uporabnika v naših implementacijah pristopov vrednotimo ločeno.

Pristop z razširitvijo za brskalnik Chrome

Naša spletna aplikacija pri uporabi pristopa z razširitvijo za brskalnik Chrome od končnega uporabnika zahteva ročno posredovanje pri naslednjih operacijah, ki so potrebne izključno za normalno delovanje zaradi podpore PHE:

a) izbira datoteke z javnim ključem uporabnika, ki se bo prenesla na strežnik banke. To mora uporabnik narediti samo enkrat in je predpogoj za izvedbo homomorfnih transakcij s strani spletne banke.

b) Klik na gumb za sprožitev postopka dekripcije z namenom prikaza trenutnega stanja na računu ali zgodovine stanj.

c) Shranjevanje javnega ključa uporabnika na prejemni strani transakcije na predvideno lokacijo v datotečnem sistemu. Slednje je potrebno pred začetkom enkripcije v primeru, da javni ključ prejemnika še ni shranjen na predvideni lokaciji, kjer ga program `HE_Client` pričakuje.

Od tega točka a) s stališča uporabniške izkušnje ni kritična, saj jo mora uporabnik praviloma izvesti le pred prvo uporabo. Točko b) bi lahko avtomatizirali in operacijo dekripcije sprožili takoj, ko se spletna stran naloži. Slednjega nismo naredili

zaradi lažjega merjenja odzivnega časa pri postopku dekripcije (predstavljeno v nadaljevanju v razdelku 6.5.2). Bolj problematična pa je točka c), saj razširitve za brskalnik Chrome ne morejo pisati na disk brez dovoljenja uporabnika. Končni uporabnik aplikacije bo zato vsakič znova (tj. za vsak javni ključ uporabnika na prejemni strani transakcije, ki le-tega še nima shranjenega lokalno) moral sam izbrati lokacijo, kamor naj se ključ shrani.

Pristop z dodatnim posvečenim strežnikom

Pri uporabi pristopa z dodatnim posvečenim strežnikom mora končni uporabnik za normalno delovanje aplikacije ročno izvajati vse operacije, navedene v prejšnjem razdelku, poleg slednjih pa mora prav tako:

a) pred prenosom svojega javnega ključa na strežnik banke le-tega najprej prenesti s posvečenega strežnika na svojo napravo,

b) prenesti javni ključ uporabnika na prejemni strani transakcije z naprave odjemalca na posvečeni strežnik.

Točka a) zopet ni kritična za uporabniško izkušnjo, saj gre za enkratno operacijo. Točka b) pa je potrebna zgolj, če javni ključ prejemnika transakcije še ni shranjen na posvečenem strežniku.

Vpletenost uporabnika za potrebe normalnega delovanja naše aplikacije bi se v obeh implementiranih pristopih dalo zmanjšati. Kljub temu pa je očitno, da je z vidika uporabniške izkušnje pristop z razširitvijo za brskalnik Chrome boljši od pristopa z dodatnim posvečenim strežnikom, saj slednji od uporabnika zahteva več ročnega posredovanja.

6.5.2 Odzivni čas

Na odzivne čase, ki jih uporabnik občuti pri uporabi spletne banke, močno vplivata dva dejavnika - izbira kriptografskih parametrov (predvsem parametra varnosti k) in pasovna širina internetne povezave med napravo odjemalca in strežnikom banke (v primeru pristopa z dodatnim posvečenim strežnikom pa tudi med napravo odjemalca in posvečenim strežnikom).

Prikaz spletnih strani s tajnopisi

Od trenutka, ko končni uporabnik v spletni banki zahteva ogled svoje domače strani (ta vsebuje tajnopis z zgodovino stanj na njegovem računu), do takrat, ko se zahtevana stran v celoti naloži v spletnem brskalniku, mora uporabnik v povprečju počakati 5 sekund. K temu času največ prispevata branje datoteke s tajnopisom na strani spletnega strežnika in prenos njene vsebine do naprave odjemalca preko omrežja. Tajnopisa končni uporabnik sicer ne vidi na spletni strani, vendar ga spletni strežnik pošlje s spletno stranjo, da ga lahko pred dekripcijo prebere razširitev za brskalnik Chrome ali pa koda v jeziku JavaScript, ki ga posreduje posvečenemu strežniku.

Prikaz trenutnega stanja na transakcijskem računu ali pregled zgodovine transakcij

Za prikaz trenutnega stanja na transakcijskem računu uporabnika oziroma za pregled zgodovine transakcij je potrebna dekripcija tajnopisa z zgodovino stanj. Pri izbranih vrednostih kriptografskih parametrov od uporabnikove zahteve za dekripcijo (tj. od klika na gumb) do prikaza čistopisa na spletni strani v povprečju preteče 12 sekund. Dekripcijo bi lahko avtomatizirali, tako da bi se le-ta samodejno sprožila takoj, ko se naloži stran. V tem primeru bi uporabnik od zahteve za ogled strani do prikaza trenutnega stanja na njegovem računu čakal 17 sekund.

Izvedba transakcije

Z vidika končnega uporabnika naše spletne banke izvedba transakcije predstavlja čas, ki preteče od trenutka, ko uporabnik na spletni strani (po vnosu zneska, namena nakazila in številke transakcijskega računa) s klikom na gumb sproži začetek transakcije, do trenutka, ko se v spletni aplikaciji pojavi obvestilo o uspešno izvedeni transakciji. Gre torej za čas izvebe transakcije, kot ga občuti uporabnik naše spletne banke. Pri izbranih vrednostih kriptografskih parametrov izvedba transakcije v povprečju traja okrog 30 sekund. Ta čas vključuje:

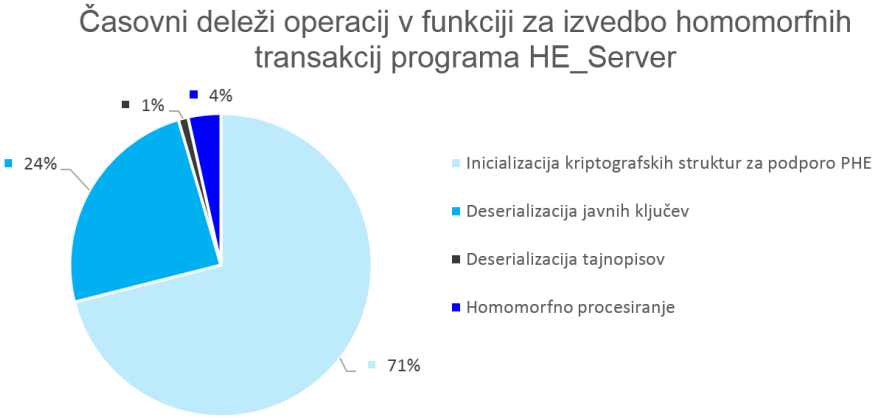
- 13.5 sekund procesiranja na strani odjemalca zaradi enkripcije zneska pred transakcijo,
- 15.5 sekund strežniškega procesiranja,

- 1–3 sekunde na račun komunikacije za prenos tajnopisov z zneskom za transakcijo na strežnik banke. Pri pristopu z dodatnim posvečenim strežnikom je ta strošek večji kot pri pristopu z razširitvijo za brskalnik Chrome zaradi predhodnega prenosa tajnopisov s posvečenega strežnika na napravo odjemalca. Klub temu slednje ni kritično, saj so zakasnitve zaradi enkripcije in strežniškega procesiranja bistveno večji.

Čas za izvedbo homomorfne transakcije na nivoju programa C++ Izvajalni čas funkcije za izvedbo homomorfnih transakcij v programu `HE_Server` (naveden zgoraj) ni enak dejanskemu času, potrebnemu za homomorfno računanje. Meritve izvajalnih časov posameznih procedur v omenjeni funkciji prikazuje diagram na sliki 6.5. Iz diagrama je razvidno, da homomorfno računanje predstavlja zgolj štiri odstotke celotnega izvajalnega časa funkcije. Ta čas, torej čas, ki ga zahteva homomorfno procesiranje v programu `HE_Server`, v nadaljevanju obravnavamo kot dejanski čas za izvedbo homomorfne transakcije. Pri izbranih vrednostih kriptografskih parametrov je naša homomorfna transakcija v povprečju 50-krat počasnejša od navadne, nehomomorfne transakcije. Kot navadno bančno transakcijo smo obravnavali par operacij seštevanja in odštevanja nad čistopisi zneskov na transakcijskih računih, izvedli pa smo jo v programski kodi v jeziku Python.

Čas za izvedbo homomorfne transakcije na nivoju spletne aplikacije Na nivoju spletne aplikacije banke je čas za izvedbo homomorfne transakcije približno enak času izvajanja celotne funkcije za izvajanje homomorfnih transakcij v programu `HE_Client` (oz. `HE_RemoteClient`) in zato traja okrog 16 sekund. Slednjo namreč pokliče funkcija v Pythonu, ki obdela odjemalčevo zahtevo za izvedbo transakcije. Za primerjavo učinkovitosti smo v podatkovnem modelu transakcije v spletni aplikaciji shranjevali tudi čistopis zneska, ki se prenaša v okviru bančne transakcije. To nam je omogočilo, da smo lahko izmerili, koliko časa traja bančna transakcija v primeru, da iz funkcije, ki v kodi Python obdela zahtevek za izvedbo transakcije, ne pokličemo funkcije programa `HE_Client` (oz. `HE_RemoteClient`), ampak samo posodobimo stanji na računih odjemalcev. Ker je poleg dejanskega časa za izvedbo transakcije na nivoju programa v C++ potrebno upoštevati še relativno velik čas za inicializacijo kriptografskih struktur in deserializacijo kritp-

tografskih ključev ter tajnopisov, je izvajanje homomorfnih bančnih transakcij na nivoju spletne aplikacije 120-krat počasnejše, kot če PHE ne bi uporabljali.



Slika 6.5: Časovni deleži operacij v funkciji za izvedbo homomorfnih transakcij programa `HE_Server`. Homomorfno procesiranje k izvajalnemu času funkcije v povprečju doprinese le 4 odstotke, kar ustreza približno 0.5 sekunde.

Trajanje operacij za podporo FHE

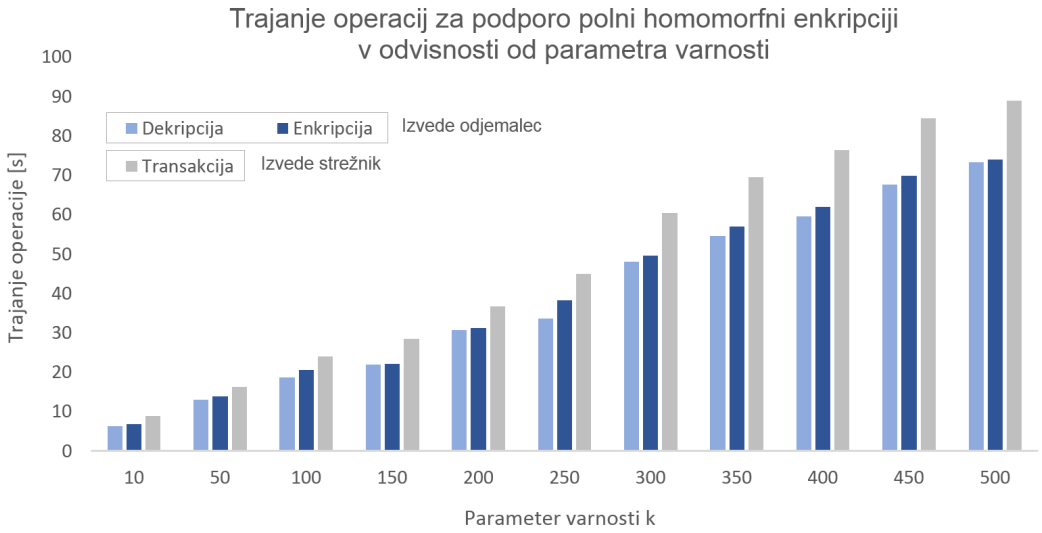
V prejšnjih razdelkih smo navedli povprečna trajanja operacij enkripcije in dekripcije (na strani odjemalca) ter bančne transakcije (na strani strežnika) pri izbranih vrednostih kriptografskih parametrov. Na tem mestu želimo poudariti, da so medsebojne razlike v trajanju omenjenih operacij relativno majhne, saj vse trajajo od 12 do 15 sekund. Slednje je v nasprotju z našimi začetnimi pričakovanji, saj smo pričakovali precej krajše postopke enkripcije in dekripcije. Kljub temu pa je dejansko trajanje programskih rutin knjižnice HElib, ki v okviru funkcij naših programov `HE_Client` in `HE_RemoteClient` izvedejo enkripcijo in dekripcijo, *bistveno manjše* od navedenih časov. Dejanski enkripcija in dekripcija v naših programih namreč prispevata le okrog 1–2 odstotka k navedenim izvajalnim časom (podobno velja tudi za dejanski čas homomorfnega procesiranja v okviru bančne transakcije, kar lahko vidimo na sliki 6.5).

Večinski delež - več kot 70 odstotkov - izvajalnih časov funkcij v naših programih `HE_Client`, `HE_RemoteClient` in `HE_Server` ne glede na želeno operacijo

(enkripcijo/dekripcijo/bačno transakcijo) je posledica inicializacije programskih struktur, ki so potrebne za normalno delovanje rutin knjižnice HElib. Pomemben delež predstavlja tudi deserializacija kriptografskega materiala. Oboje je nujno potrebno pred izvedbo enkripcije, dekripcije in pred homomornfim procesiranjem ter predstavlja ozko grlo naših programskih rešitev, saj močno vpliva na odzivne čase v spletni aplikaciji in s tem na transparentnost delovanja za končnega uporabnika.

Na izvajalne čase funkcij za enkripcijo, dekripcijo in izvedbo homomorfnih transakcij močno vpliva parameter varnosti k. S povečevanjem parametra varnosti je opazno daljši predvsem čas, potreben za inicializacijo programskih struktur, ki prispeva večinski delež k skupnemu izvajalnemu času. Trajanja funkcij za enkripcijo, dekripcijo in homomorfne transakcije pri različnih vrednosti parametra varnosti prikazuje slika 6.6. Iz slike je razvidno, da so trajanja omenjenih operacij premo sorazmerna s parametrom varnosti k, pri čemer trajanje transakcij s povečevanjem parametra k narašča hitreje kot trajanje enkripcije in dekripcije.

Na podlagi pregleda vpletenosti uporabnika in odzivnega časa pri obeh implementiranih pristopih na strani odjemalca lahko zaključimo, da je uporaba pristopa z razširitvijo za brskalnik Chrome bolj transparentna za končnega uporabnika. Ker pristop z dodatnim posvečenim strežnikom zahteva komunikacijo odjemalca z dodatno komponento, so odzivni časi daljši, prav tako pa je delo s spletno aplikacijo za uporabnika bolj zamudno, saj od njega zahteva več ročnega posredovanja.



Slika 6.6: Trajanje operacij enkripcije in dekripcije na strani odjemalca in transakcij na strani strežnika v odvisnosti od parametra varnosti k. Ker operacija enkripcije pred transakcijo zajema branje in deserializacijo dveh javnih ključev, za tem pa še dve enkripciji operanda z ločenima javnima ključema, je vedno malo dražja od dekripcije.

Poglavje 7

Sklep

Zasebna obdelava občutljivih podatkov v zaupanja nevrednih okoljih računalniških oblakov je težek problem, za katerega danes ni splošne in dobro uveljavljene rešitve. V pričujočem delu smo preučili nekaj trenutno najbolj aktualnih pristopov, ki oblačnim aplikacijam obljubljajo višjo raven zasebnosti med obdelavo podatkov, kot je danes običajno. S pomočjo programske knjižnice HElib smo na primeru preproste spletne banke implementirali enega izmed preučenih pristopov - polno homomorfno enkripcijo. Naša implementacija sledi modelu odjemalec-strežnik in omogoča izvajanje t. i. *homomorfnih bančnih transakcij*, pri katerih strežnik obdeluje tajnopise stanj na transakcijskih računih strank. Ker se strežnik ne zaveda vsebine podatkov, ki jih obdeluje, naša spletna banka ohranja zasebnost bančnih transakcij. Na strani odjemalca smo za enkripcijo podatkov in dekripcijo tajnopisov predlagali tri različne alternative, od katerih smo dve, pristop z razširitvjo za brskalnik Chrome in pristop z dodatnim posvečenim strežnikom, uspešno realizirali. Definirali smo tudi mere, na podlagi katerih smo nazadnje ovrednotili praktično uporabnost naših implementacij.

Ugotovili smo, da je ob uporabi izbrane programske knjižnice za PHE režijskega dela na strani odjemalca, ki je potrebno za podporo operacijama enkripcije in dekripcije, bistveno več, kot smo pričakovali. Podobno velja tudi za strežniški del, kjer smo na nivoju programa v C++ izmerili presenetljivo majhen prispevek dejanskega homomorfnega računanja k skupnemu izvajalnemu času. Kot ozko grlo naših implementacij smo identificirali postopek inicializacije programskih struktur, ki so potrebne za enkripcijo, dekripcijo in homomorfno procesiranje. S tem

smo pokazali, da neučinkovitost homomorfnega procesiranja na strani strežnika, s katerim se tradicionalno ukvarjata tako teorija kot tudi praksa na področju PHE, ni edina ovira, ki preprečuje njeno širšo uveljavitev v praksi.

Na primeru naše spletne aplikacije smo lahko videli, da je uporaba PHE za izvedbo operacij, ki vplivajo na zasebne podatke več kot enega odjemalca, zaradi prenašanja zelo velikih kriptografskih ključev po omrežju nerodna.

Prav tako smo prišli do spoznanja, da je s trenutno razpoložljivimi programskimi orodji implementacija podpore za PHE v spletnih aplikacijah zamudna, produkcijska zrelost trenutno najbolj napredne knjižnice za njeno realizacijo pa je nizka in ne omogoča izgradnje poljubnih, praktično uporabnih aplikacij. Pričakujemo, da bodo bodoče nadgradnje omenjene programske knjižnice odpravile številne pomanjkljivosti, kar bo pozitivno vplivalo na nivo njene praktične uporabnosti. Kljub temu pa menimo, da je za uveljavitev polne homomorfne enkripcije v praksi ključnega pomena tudi obstoj razvijalcem prijaznega programskega vmesnika v katerem od visokonivojskih programskih jezikov, ki v trenutku pisanja tega besedila ne obstaja.

Možnosti za izboljšave in nadaljnje delo je veliko. Na primer, za povečanje transparentnosti uporabe PHE za končnega uporabnika in zmanjšanje porabe računskih virov bi bilo smiselno optimizirati izvajalne čase funkcij programov v C++. Ponovna uporaba programskih struktur, ki so rezultat drage inicializacije, vendar so pri izbranem naboru kriptografskih parametrov vedno enake, bi lahko prinesla bistveno krajše povprečne izvajalne čase.

Poleg tega bi z bolj pazljivim načrtovanjem pristopov na strani odjemalca lahko razvili rešitve, ki niso tako tesno sklopljene s spletno aplikacijo banke, in s tem dosegli njihovo večjo uporabnost in ponovno uporabljivost. Slednje bi lahko realizirali pri pristopu z dodatnim posvečenim strežnikom, če bi le-ta nastopal v vlogi posrednika med odjemalcem in strežnikom spletne banke in ne bi komuniciral zgolj z odjemalcem.

Kljub enostavni aplikacijski logiki in številnim predpostavkam, ki jih naredimo v delu, pa naše praktično delo predstavlja prvo nam znano celovito spletno rešitev za podporo PHE, ki upošteva tako vlogo lahkega odjemalca kot tudi strežnika v računalniškem oblaku. Zgodnje implementacije PHE v realnih postavitvah se bodo najverjetneje soočale s podobnimi težavami kot naše programske rešitve za dokaz

izvedljivosti. Prav tako menimo, da so naši predlogi za implementacijo podpore PHE na strani odjemalca splošni, in pričakujemo, da bodo zgodnje implementacije uporabljale podobne rešitve, kot jih predlagamo v pričujočem delu.

Čeprav so mnenja strokovnjakov glede prihodnosti praktične uporabe PHE deljena, napredek na tem področju vsekakor ostaja tako v interesu odjemalcev kot tudi ponudnikov oblačnih storitev. Posledice, ki jih ima uporaba PHE v oblačnih aplikacijah, so lahko za oboje izjemnega pomena. Ker PHE varuje odjemalčeve podatke tudi med obdelavo, je zanj tveganje razkritja zaradi nepazljivega ali zlonamernega ravnanja ponudnika oblačnih storitev manjše. Obenem pa uporaba PHE zmanjšuje poslovno škodo za ponudnika storitev ob morebitnih napadih od zunaj. Kljub trenutni nedosegljivosti tako praktična uporaba PHE zaradi vseh omenjenih prednosti ostaja aktualen problem.

Literatura

[1] R. Agrawal, J. Kiernan, R. Srikant in Y. Xu, "Order preserving encryption for numeric data", v zborniku *Proceedings of the 2004 ACM SIGMOD international conference on Management of data*, 2004, str. 563–574.

[2] J. Bacon, D. Evans, D. M. Eyers, M. Migliavacca, P. Pietzuch in B. Shand, "Enforcing end-to-end application security in the cloud", v zborniku *Middleware 2010*, 2010, str. 293–312.

[3] V. Balhara, "Cloud security: Theory and practice", *IJCA Proceedings on 4th International IT Summit Confluence 2013 - The Next Generation Information Technology Summit*, št. Confluence 2013, zv. 3, str. 14–17, 2014.

[4] D. Bogdanov, R. Talviste in J. Willemson, "Deploying secure multi-party computation for financial data analysis", v zborniku *International Conference on Financial Cryptography and Data Security*, 2012, str. 57–64.

[5] A. Boldyreva, N. Chenette in A. O'Neill, "Order-preserving encryption revisited: Improved security analysis and alternative solutions", v zborniku *Annual Cryptology Conference*, 2011, str. 578–595.

[6] (2016) Boost C++ library - Boost.Python. Dostopno na: http://www.boost.org/doc/libs/1_61_0/libs/python/doc/html/index.html (pridobljeno 18.8.2016).

[7] (2015) Boost C++ library - Serialization. Dostopno na: http://www.boost.org/doc/libs/1_36_0/libs/serialization/doc/index.html (pridobljeno 31.8.2016).

[8] Z. Chen, J. Wang, Z. Zhang in X. Song, "A fully homomorphic encryption scheme with better key size", *China Communications*, št. 11, zv. 9, str. 82–92, 2014.

[9] C. Devet, "Evaluating private information retrieval on the cloud", *University of Waterloo*, št. 1, zv. 1, 2013.

[10] W. Du in M. J. Atallah, "Secure multi-party computation problems and their applications: A review and open problems", v zborniku *Proceedings of the 2001 Workshop on New Security Paradigms*, ser. NSPW '01, 2001, str. 13–22.

[11] L. Ducas in D. Micciancio, "FHEW: bootstrapping homomorphic encryption in less than a second", v zborniku *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, 2015, str. 617–640.

[12] C. Gentry, "A fully homomorphic encryption scheme", doktorska disertacija, Stanford University, 2009.

[13] (2016) GMP - The GNU Multiple Precision Arithmetic Library. Dostopno na: https://gmplib.org/ (pridobljeno 25.8.2016).

[14] S. Goldwasser, "Multi party computations: past and present", v zborniku *Proceedings of the sixteenth annual ACM symposium on Principles of distributed computing*, 1997, str. 1–6.

[15] (2016) Google Chrome Extensions - Content Scripts. Dostopno na: https://developer.chrome.com/extensions/content_scripts (pridobljeno 19.8.2016).

[16] (2016) Google Chrome Extensions - Event Pages. Dostopno na: https://developer.chrome.com/extensions/event_pages (pridobljeno 19.8.2016).

[17] (2016) Google Chrome Extensions - Native Messaging. Dostopno na: https://developer.chrome.com/extensions/nativeMessaging (pridobljeno 31.8.2016).

[18] (2016) Google Chrome Extensions - What are extensions? Dostopno na: https://developer.chrome.com/extensions (pridobljeno 19.8.2016).

[19] U. Greveler, B. Justus in D. Loehr, "A privacy preserving system for cloud computing", v zborniku *IEEE 11th International Conference on Computer and Information Technology (CIT)*, 2011, str. 648–653.

[20] M. Grinberg, *Flask Web Development: Developing Web Applications with Python*. "O'Reilly Media, Inc.", 2014.

[21] S. Halevi in V. Shoup, "Design and implementation of a homomorphic-encryption library", *IBM Research (Manuscript)*, 2013.

[22] S. Halevi in V. Shoup, "Algorithms in HElib", v zborniku *Advances in Cryptology–CRYPTO 2014*, 2014, str. 554–571.

[23] S. Halevi in V. Shoup, "Bootstrapping for HElib", v zborniku *Advances in Cryptology–EUROCRYPT 2015*, 2015, str. 641–670.

[24] (2016) HElib - An implementation of homomorphic encryption. Dostopno na: https://github.com/shaih/HElib (pridobljeno 25.8.2016).

[25] Y. Huang in I. Goldberg, "Outsourced private information retrieval", v zborniku *Proceedings of the 12th ACM workshop on Workshop on privacy in the electronic society*, 2013, str. 119–130.

[26] (2016) ITRC Data Breaches Report 2016. Dostopno na: http://www.idtheftcenter.org/images/breach/ITRCBreachReport2016.pdf (pridobljeno 10.9.2016).

[27] F. Kerschbaum, "Privacy technologies and policy: First annual privacy forum, apf 2012, limassol, cyprus, october 10-11, 2012, revised selected papers", B. Preneel in D. Ikonomou, Eds., Berlin, Heidelberg, 2014, str. 41–54.

[28] V. Lyubashevsky, C. Peikert in O. Regev, "On ideal lattices and learning with errors over rings", v zborniku *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, 2010, str. 1–23.

[29] P. Mell in T. Grance, "The NIST definition of cloud computing", *Communications of the ACM*, št. 53, zv. 6, str. 50, 2010.

[30] M. Naehrig, K. Lauter in V. Vaikuntanathan, "Can homomorphic encryption be practical?" v zborniku *Proceedings of the 3rd ACM workshop on Cloud computing security workshop*, 2011, str. 113–124.

[31] (2016) New Relic - Application Performance Monitoring & Management. Dostopno na: https://newrelic.com/ (pridobljeno 25.8.2016).

[32] (2016) NTL : A Library for doing Number Theroy. Dostopno na: http://www.shoup.net/ntl/ (pridobljeno 25.8.2016).

[33] (2016) Openstack - open source software for creating private and public clouds. Dostopno na: https://www.openstack.org/ (pridobljeno 8.9.2016).

[34] A. Page, O. Kocabas, S. Ames, M. Venkitasubramaniam in T. Soyata, "Cloud-based secure health monitoring: Optimizing fully-homomorphic encryption for streaming algorithms", v zborniku *2014 IEEE Globecom Workshops (GC Wkshps)*, 2014, str. 48–52.

[35] (2016) RightScale - State of the Cloud Report. Dostopno na: https://www.rightscale.com/lp/state-of-the-cloud (pridobljeno 14.8.2016).

[36] M. D. Ryan, "Cloud computing security: The scientific challenge, and a survey of solutions", *Journal of Systems and Software*, št. 86, zv. 9, str. 2263–2268, 2013.

[37] (2015) Hcrypt project - Scarab library. Dostopno na: http://shapecpu.de/scarab-library/ (pridobljeno 18.8.2016).

[38] (2016) Skyhigh Networks - Cloud Adoption and Risk Report. Dostopno na: https://www.skyhighnetworks.com/cloud-report (pridobljeno 14.8.2016).

[39] D. Song, E. Shi, I. Fischer in U. Shankar, "Cloud data protection for the masses", *Computer*, zv. 1, str. 39–45, 2012.

[40] H. Takabi, J. B. Joshi in G.-J. Ahn, "Security and privacy challenges in cloud computing environments", *IEEE Security & Privacy*, zv. 6, str. 24–31, 2010.

[41] M. Tebaa in S. E. Hajji, "Secure cloud computing through homomorphic encryption", *arXiv preprint arXiv:1409.0829*, 2014.

[42] M. Van Dijk in A. Juels, "On the impossibility of cryptography alone for privacy-preserving cloud computing", *HotSec*, št. 10, str. 1–8, 2010.

[43] (2016) W3Schools - OS Statistics. Dostopno na: http://www.w3schools.com/browsers/browsers_os.asp (pridobljeno 19.8.2016).

[44] B. Yee, D. Sehr, G. Dardyk, J. B. Chen, R. Muth, T. Ormandy, S. Okasaka, N. Narula in N. Fullagar, "Native client: A sandbox for portable, untrusted x86 native code", v zborniku *30th IEEE Symposium on Security and Privacy*, 2009, str. 79–93.

[45] A. Zakai, "Emscripten: an LLVM-to-JavaScript compiler", v zborniku *Proceedings of the ACM international conference companion on Object oriented programming systems languages and applications companion*, 2011, str. 301–312.

[46] D. Zissis in D. Lekkas, "Addressing cloud computing security issues", *Future Generation computer systems*, št. 28, zv. 3, str. 583–592, 2012.